
Pro gradu -tutkielma

Maantiede

Geoinformatiikka

# Liikennemerkkien automaattinen tunnistaminen panoraamakuvilta

Olli Rantanen
2020

Ohjaajat:
Petteri Muukkonen
Antti Jakobsson
Teemu Mielonen
Juha Oksanen

HELSINGIN YLIOPISTO
MATEMAATTIS-LUONNONTIETEELLINEN TIEDEKUNTA
GEOTIETEIDEN JA MAANTIETEEN OSASTO
MAANTIEDE

Pl 64 (Gustaf Hällströmin katu 2)
00014 Helsingin yliopisto

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

| Tiedekunta/Osasto Fakultet/Sektion – Faculty | Laitos/Institution– Department | |
|---|---|---|
| Matemaattis-luonnontieteellinen | Geotieteiden ja maantieteen osasto | |
| Tekijä/Författare – Author | | |
| Olli Rantanen | | |
| Työn nimi / Arbetets titel – Title | | |
| Liikennemerkkien automaattinen tunnistaminen panoraamakuvilta | | |
| Oppiaine /Läroämne – Subject | | |
| Maantiede - Geoinformatiikka | | |
| Työn laji/Arbetets art – Level | Aika/Datum – Toukokuu 2020 | Sivumäärä/ Sidoantal – Number of pages 76 |
| Pro gradu -tutkielma | | |


Tiivistelmä

Uuden tieliikennelain mukanaan kunnille tuomat velvoitteet, kuten liikenteenohjaukseen käytetyn välineistön (esim. liikennemerkkien) ylläpitovastuu, siirtyy kunnille 1.6.2020. Kenttäinventoimalla suoritettava liikennemerkkien kunnon ja sijainnin selvittäminen on usein työlästä ja tuottaa kustannuksia. Tässä tutkimuksessa selvitetään, miten näitä voidaan automatisoidusti inventoida panoraamakuvilta. Samalla verrataan panoraamakuvien ja niistä luotujen osakokonaisuuksien (pilkottujen kuvien) soveltuvuutta kyseiseen tarkoitukseen. Tunnistuksen tuloksena syntyviä havaintoja verrataan Väyläviraston ylläpitämään avoimeen liikennemerkkiaineistoon sekä tunnistettujen kohteiden sijainti lasketaan kuvilta. Työssä tutustutaan myös eri kohteentunnistusalgoritmien toimintaan sekä selvitetään, miten liikennemerkkien automaattisessa tunnistuksessa on onnistuttu muissa tutkimuksissa.

Aineistona toimii Inkoosta otettujen panoraamakuvien lisäksi Mapillaryn toimittamat kuva-aineistot, joita käytetään YOLOv3- kohteentunnistusalgoritmin koulutukseen ja testaukseen. Työssä esitellään myös YOLOv3-koulutuksen toteuttaminen ja käydään läpi tarvittavat ohjelmistot sen implementoinnissa toiseen työhön. Koulutus vaatii riittävän GPU:n lisäksi erilaisia ohjelmia sekä runsaasti kuva-aineistoa, jotta ylisovittamisen riskiltä vältytään.

Tulosten perusteella pilkotut kuvat tuottavat paremman tuloksen verrattuna panoramakuviin. Pilkotuilta kuvilta jokainen ajoreitin varrella ollut kärkikolmio tunnistettiin, kun taas panoraamakuvilta tämä ei onnistunut. Lisäksi algoritmin kyky sijoittaa kärkikolmion sijainti kuvalle oli varsin hyvä saavuttaen keskimäärin IoU-arvon 0,86, kun se panoraamoilla oli 0,52. Samoin tulosten luotettavuutta kuvaavat *Precision*- ja *Recall*-arvot olivat huomattavasti korkeammat kuin panoraamakuvilla. Työssä havaittiin lisäksi, että Väyläviraston avoimesta aineistosta puuttuu useita kärkikolmioita. Kuvilta onnistuttiin myös laskemaan muutaman metrin tarkkuudella kärkikolmioiden sijainti maastossa.

Tutkimuksen perusteella kohteentunnistusalgoritmit tuottavat merkittävää hyötyä kohteiden automaattisessa tunnistuksessa. Algoritmien hyödyntämistä tulevaisuudessa mahdollistaa lisääntyvä kuva-aineistojen määrä sekä laskentatehon kasvu. Hyödyntämällä kohteentunnistusalgoritmeja kuntien on mahdollista helpottaa uuden tieliikennelain velvoitteiden noudattamista. Tämän myötä algoritmien suosio voi kasvaa tulevaisuudessa. Kohteentunnistusalgoritmien implementointiin tarvitaan kuitenkin ohjeistusta ja käyttötapauksia, joita tämä tutkimus tuloksillaan edistää.



Avainsanat – Nyckelord – Keywords
Kohteentunnistus, Koneoppiminen, Liikennemerkit, Neuroverkot, Panoraamakuva, Tieaineistot, YOLO

Säilytyspaikka – Förvaringställe – Where deposited
E-thesis

Muita tietoja – Övriga uppgifter – Additional information
Työ toteutettiin Maanmittauslaitoksen toimeksiantona

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

| Tiedekunta/Osasto Fakultet/Sektion – Faculty | Laitos/Institution– Department | |
|---|---|---|
| Faculty of Science | Department of Geosciences and Geography | |
| Tekijä/Författare – Author | | |
| Olli Rantanen | | |
| Työn nimi / Arbetets titel – Title | | |
| Detecting traffic signs from panoramic pictures | | |
| Oppiaine /Läroämne – Subject | | |
| Geography - Geoinformatics | | |
| Työn laji/Arbetets art – Level | Aika/Datum – /May 2020 | Sivumäärä/ Sidoantal – Number of pages |
| Master's thesis | | 76 |


Abstract

The new legislation that is based on the Road Traffic Act comes with new obligations to municipalities. These relate to many different aspects, for example municipalities are mandated to collect data from their traffic signs. Collecting these with in-field observations and inventories, is costly and labor intensive. this study explores how these could be collected automatically from panoramic pictures and how adequate panoramas themselves are for the task. The panoramas are also sliced to smaller subsets and the detection quality is compared between these two datasets. Also, the detections are compared to existing FTIA (Finnish Transport Infrastructure Agency) traffic sign dataset. Furthermore, the locations of the detected signs are calculated from the photos. The study also dives into the deep end with detail inspection of the current object detection algorithms and discuss how other studies, which have examined traffic sign detections, compare to our results.

The data in the study is based on panoramic photos 1) taken from Inkoo municipality and 2) pictures delivered by Mapillary. These are used to train and test YOLOv3-object detection algorithm. The study also discusses the programs and requirements to implement YOLO to another project. The training is compute-intensive and requires powerful GPU as well as considerable amount of training data. The risk of overfitting is immanent with object detection and therefore the training data should have enough variation.

The results sections discuss that panorama subsets are indeed better suited for the task. Among the subsets, all the yield signs along the routes were discovered and the detection accuracy was significantly better than with panoramas. Common metric to evaluate detection quality is to use Intersection over Union (IoU) to detected objects. IoU-value with subsets was on average 0.86 and with panoramas 0.52, hence signaling much better performance using smaller photos for the task. Precision and recall were also calculated for these two data sets, which indicated that subsets are better suited for the task.
The FTIA's public data set will be compared to the detections made on the study. On this comparison it is found out that some yield signs were missing. Calculating the locations of the yield signs proved to be difficult and the visually expressed results were not very accurate.

Based on this study, it can be safely said that it is beneficial to use object detection algorithms to automatically detect traffic signs. The amount of data and the computing power keeps rising in the future which improves the detection algorithms simultaneously. Using said algorithms the municipalities can benefit and be better prepared for the new legislation. Implementing an object detection algorithm is not an easy task so guidance and use cases, as this study, are needed.



Avainsanat – Nyckelord – Keywords
Machine learning, Neural networks, Object detection, Panoramic pictures, Road dataset, Traffic signs, YOLO

Säilytyspaikka – Förvaringställe – Where deposited
E-thesis

Muita tietoja – Övriga uppgifter – Additional information
The study was commissioned by the National Land Survey of Finland.

# Sisällysluettelo

# Avainsanat

Annotoitu kuva = kuva, jolle on luotu tunnus, josta selviää bounding box -rajaus koulutettavasta kohteesta

AP = Average Precision

Bounding box = suorakulmarajaus

CNN = Convolutional Neural Network

Luottamustaso (*confidence level*) = tunnistuksen luotettavuuden arvio

CPU = Central Processing Unit

Darknet = avoimen lähdekoodin neuraalisten verkkojen viitekehys, jota käytetään YOLO-verkoston implementointiin

Digiroad = kansallinen tie- ja katuverkon tietojärjestelmä

GNSS = Global Navigation Satellite System

GPU = Graphics Processing Unit

FC = Fully connected (taso tai verkosto)

FPS = kuvataajuus (Frames Per Second)

IMU= Inertial Measurement Unit

IoU = Intersection over Union, leikkauksen ja unionin osamäärä

mAP = Mean Average Precision

Mapillary = yksityinen kaupallinen yritys, jonka alustalle vapaaehtoiset toimittavat kuva-aineistoa yleensä katutasonnäkymistä.

MML = Maanmittauslaitos

NMS = Non-Maximum suppression

Pano-RSOD = Panoramic road scene object detection-aineisto

Painoarvotiedosto (weights-file) = koulutuksen pohjalta syntyvä koulutettu verkosto

Pooling = yhdistämistaso

RoI = Region of Interest

R-CNN = Convolutional Neural Network with Region proposals

Kynnysarvo (*Threshold value*) = kohteentunnistusalgoritmissa asetettava minimitunnistustaso

SSD = Single Shot MultiBox Detector

YOLO = You Only Look Once

# 1. Johdanto

## 1.1 Tutkimuksen tausta

Uusi tieliikennelaki (729/2018) astuu voimaan 01.06.2020, ja sen myötä kunnille siirtyy vastuu toimittaa tietoja esimerkiksi liikennemerkeistään paikkatietoaineistona Väyläviraston Digiroad-järjestelmään. Digiroad on kansallinen tie- ja katuverkon tietojärjestelmä, johon on koottu avoimeksi ja ilmaiseksi saataville erinäistä tiestöön liittyvää tietoa. Uuden lain myötä kuntien pitää löytää keinoja koota tehokkaasti liikennemerkeistään paikkatietoaineistoa, jotta ne vastaavat lain tuomaan velvoitteeseen. Näiden automaattinen tunnistaminen ja sijainnin laskeminen nopeuttaisi huomattavasti tätä työtä verrattuna havaintojen manuaaliseen keräämiseen. Hienonen (2014) toteaa liikennemerkkien automaattisen inventoinnin ja sitä kautta muodostuvan tietokannan avulla voitavan esimerkiksi säästää kustannuksissa. Lisäksi liikenneturvallisuus parantuisi, tiedon hallinnointi helpottuisi älykkäissä ajoratkaisuissa sekä merkkien sijaintitietoa voisi parantaa (Hienonen 2014).

Liikennemerkkien automaattisen tunnistuksen todetaan olevan myös yksi tärkeimmistä keinoista autonomisten ajoneuvojen toiminnan mahdollistamiseksi (Temel et al. 2019). Temel et al. (2019) jatkavat autonomisten ajoneuvojen luotettavuuden pohjautuvan teknologioihin, jotka prosessoivat ja analysoivat sensoreilta saatavaa informaatiota tarkasti. Samoin Li et al. (2019) puhuvat kohteiden tunnistuksen olevan vaatimuksena autonomisten ajoneuvojen kehitykselle, jotta esimerkiksi jalankulkija ja muut autot havaitaan ympäriltä.

Suomessa Väylävirasto on tutkinut mahdollisuuksia hyödyntää konenäköä tieomaisuuden hallinnassa (Suuriniemi et al. 2019). Tutkimuksen mukaan konenäön hyödyntäminen olisi mahdollista, aineistoa pitää olla riittävästi. Yleisesti aineiston määrän arvioidaan kasvavan tulevaisuudessa jatkuvasti, mikä auttanee tässä. Toisaalta aineiston määrän kasvu voi johtaa automatiikan implementoinnin tarpeellisuuteen, koska aineistojen analysointi ihmisten toimesta on usein liian työlästä (Suuriniemi et al. 2019). Kohteentunnistusalgoritmien hyödyntämisessä liikennemerkkien ja muiden kohteiden tunnistamisessa on päästy hyviin lopputuloksiin (Arcos-García et al. 2018, Hienonen 2014, Stallkamp et al. 2012, Voulodimos et al. 2017, Zhang et al. 2017).

Algoritmien käyttöön ja koulutukseen liittyy myös useampia haasteita. Esimerkiksi kuvausolosuhteisiin vaikuttavat ongelmat, kuten sade ja sumuisuus, olisi otettava paremmin huomioon kuvauksessa (Hienonen 2014, Li et al. 2019, Stallkamp et al. 2012, Temel et al. 2019, Zhu et al. 2016). Myöskään toimivaa viitekehystä, jonka pohjalta oikeanlaisen koneoppimisalgoritmin valinta tietyn aineiston perustella olisi optimaalista, ei ole vielä luotu. Siten perusteet jonkin menetelmän poissulkemiseenkin ovat puutteelliset (Voulodimos et al. 2017).

Kansainväliset tutkimukset osoittavat edelleen puutteita myös algoritmien toiminnassa ja niiden antamien tulosten vertailussa. Algoritmien luontia ovat ohjanneet julkaistut aineistot, joiden on arvioitu useammassa tutkimuksessa olevan epäpäteviä reaalimaailman haasteisiin (Zhu et al. 2016, Wu et al. 2019). Nämä yleensä koostuvat pienemmistä kuvista, joissa tunnistettavat kohteet täyttävät kuvasta huomattavan osan, jolloin ne on helpompi havaita. Reaalimaailmassa kohteet voivat kuitenkin olla hyvin pieniä, mikä korostuu erityisesti panoraamakuvissa. Tulevaisuudessa olisikin oleellista luoda paremmin tähän vastaavia algoritmeja (Li et al. 2019, Zhu et al. 2016). Vaikka panoraamakuvilta on pyritty tunnistamaan muissakin tutkimuksissa kohteita (esim. Li et al. 2019, Zhu et al. 2016), ei vertailua panoraamakuvilta tunnistamisen ja niistä pilkottujen kuvien välillä ei aiheeseen tutuessa löytynyt. Niinpä onkin mielenkiintoista tutustua miten näiden välillä tunnistustulokset eroavat.

## 1.2 Tutkimuskysymykset ja tavoitteet

Paikkatietoaineistojen tuottaminen ja sisällön tarkentaminen uusista aineistolähteistä ja tietotekniikan kehitystä hyödyntäen on luonut uusia mahdollisuuksia viime vuosina. Esimerkiksi Guo et al. (2020) käyvät läpi useampia spatiaalisia aineistoja hyödyntäviä tekoälysovellutuksia, kuten ilmakuvilta tehtävää automaattista maanpeitteen luokittelua sekä maanpinnanmuotojen mallintamista. Myös kohteentunnistusalgoritmeja on käytetty onnistuneesti tieaineistojen päivityksessä (esim. Hienonen 2014) ja tässä työssä haluttiin selvittää, miten tunnistaminen onnistuu panoraamakuvilta. Tämän lisäksi vertaillaan miten tunnistuksen tulokset muuttuvat, kun tunnistuksessa käytetään panoraamoista luotuja osakokonaisuuksia (pilkottuja kuvia) panoraamojen sijaan.

Tunnistuksen tuloksena syntyviä havaintoja verrataan Väyläviraston ylläpitämään avoimeen liikennemerkkiaineistoon sekä tunnistettujen kohteiden sijainti lasketaan kuvilta. Työssä tutustutaan myös eri koneoppimisalgoritmien toimintaan sekä selvitetään, miten liikennemerkkien automaattisessa tunnistuksessa on onnistuttu muissa tutkimuksissa. Tutkimus toteutettiin Maanmittauslaitoksen toimek-

siantona ja pyrkimyksenä on löytää keino vähentää tarvetta maastokäynteihin sekä kenttäinventointiin, mikä säästää kuluja. Parhaassa tapauksessa kuvia voisi ottaa muun liikenteen toimesta, kuten säännöllistä aikataulua kulkevien bussien avulla, jolloin kuva-aineistoa ei tarvitsisi erikseen kerätä. Osa tieverkosta jäisi tällöin todennäköisesti kuvaamatta, mutta kohdistamalla kuvaukset vain näihin osuuksiin kuluja syntyisi vähemmän.

Kansainvälisesti liikennemerkeistä löytyy monipuolisesti avointa aineistoa erilaisista tietolähteistä. Suomesta ei kuitenkaan ollut julkista aineistoa saatavilla. Tässä tutkimuksessa aineistona hyödynnettiin Mapillaryn kuvia sekä Inkoosta otettuja panoraamakuvia. Suuri osa Mapillaryn kuva-aineistosta on tuotettu joukkoistamalla yksityishenkilöiden ja yksityisten toimijoiden puolesta. Hyödyntämällä tämänkaltaisia valmiita aineistoja viranomaisten tarve järjestää koko Suomen kattavia kuvauksia ei olisi välttämätöntä.

Tutkimuksessa pyritään vastaamaan seuraaviin kysymyksiin:

1. Onnistuuko kohteiden tunnistaminen panoraamakuvilta?
2. Mikä on kuvan koon merkitys tunnistuksessa ja sen tarkkuudessa?
3. Miten kohteiden sijainti voidaan määrittää YOLO:n tunnistamien *bounding box* -rajausten perusteella?
4. Mikä on Väyläviraston valtakunnallisen kärkikolmioaineiston laatu verrattuna omiin havaintoihin?

Hypoteesina on, että pilkotuilta kuvilta tunnistus tulee onnistumaan paremmin kuin panoraamakuvilta, kohteiden sijainti onnistutaan selvittämään sekä kärkikolmioita tullaan havaitsemaan uusissa sijainneissa. Tavoitteena on tunnistaa kärkikolmioita kuvilta tarkasti sekä tehokkaasti. Lisäksi pyritään selittämään prosessi algoritmin toiminnan taustalla sekä antamaan tarvittavat tiedot algoritmin implementointiin toisessa työssä.

# 2. Tausta

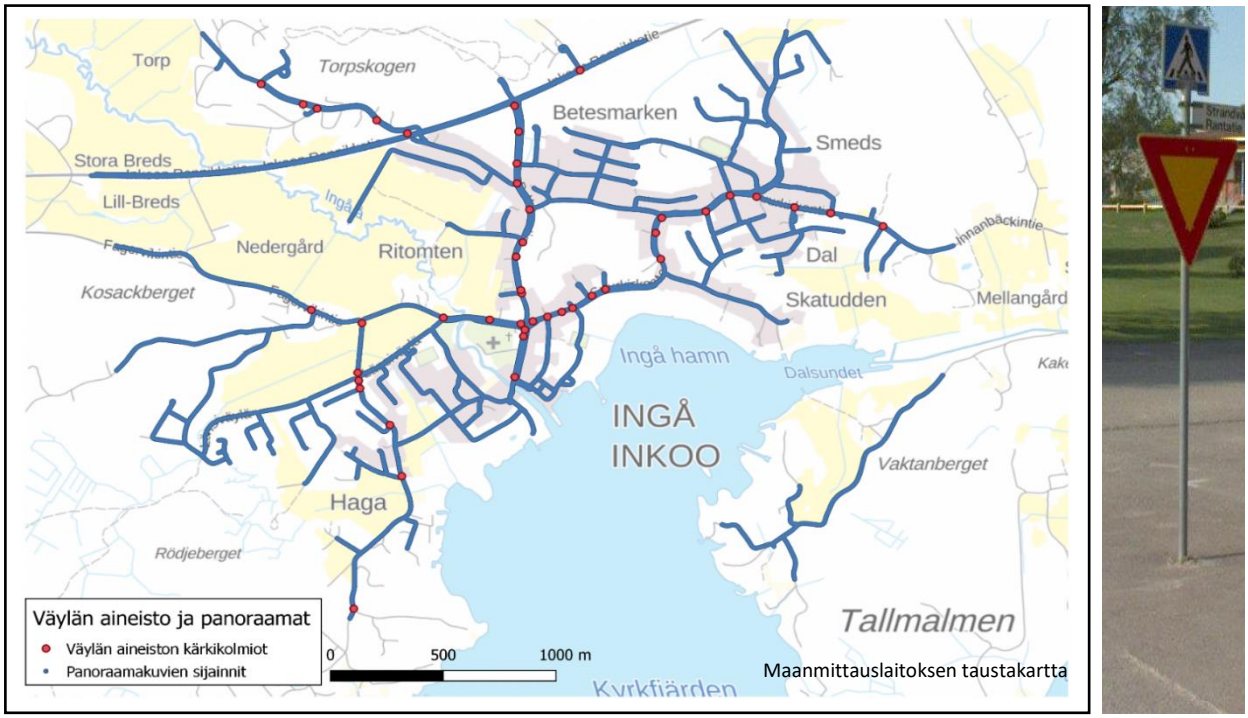
## 2.1 Liikennemerkit väyläomaisuutena

Liikennemerkit ovat olennainen osa Digiroad-tietojärjestelmää, johon kerätään tietoa koko Suomen alueelta. Uusi tieliikennelaki (729/2018) astuu voimaan 1.6.2020, minkä myötä kunnille siirtyy velvollisuus ilmoittaa tietystä väyläomaisuudesta (kuten liikennemerkeistä) tiedot Digiroad-järjestelmään. Digiroad on laaja-alaisesti käytössä monella paikkatietoalan toimijalla ja monessa järjestelmässä, joten sen tietojen oikeellisuus on erittäin tärkeää. Liikennemerkkien käyttöä ohjaa Suomessa Väylävirasto.

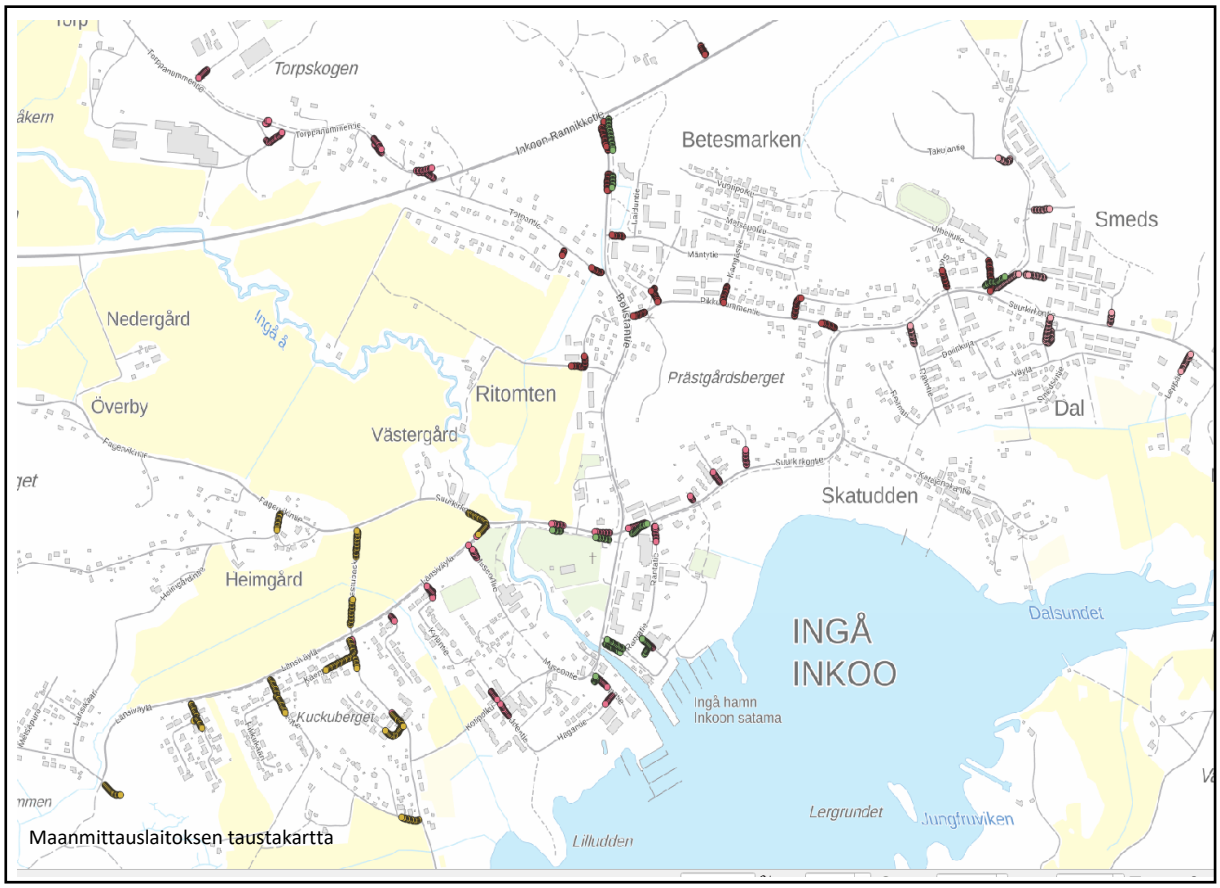
Liikennemerkkien tarkoitus on antaa tienkäyttäjälle tietoa oikeanlaisesta käyttäytymisestä liikenteessä. Pyrkimyksenä on, että merkit olisivat mahdollisimman yksinkertaisia sekä helposti havaittavia liikkuvasta ajoneuvosta, jotta niihin voi reagoida mahdollisimman nopeasti (Hienonen 2014, Velhonoja & Karhunen 2003, Stallkamp et al. 2012, 17). Liikennemerkeillä on kolme eri kokoluokkaa ja yleisintä on käyttää keskikoisia merkkejä. Pieniä merkkejä käytetään usein kevyen liikenteen väylillä ja suuria moottoritieliikenteessä (Velhonoja & Karhunen 2003, 19).

Liikennemerkkien värejä ohjaa kansainväliset asetukset ja niissä suositaan helposti erottuvia värejä. Erityispiirteenä Suomessa käytetään keltaista usein valkoisen sijaan, jotta sääolosuhteet (kuten lumi) eivät häiritse näkyvyyttä (Hienonen 2014, 15). Merkkien näkyvyyteen vaikuttavat yleisesti kuvaamisessa havaittavat ongelmat, kuten valon määrän vaihtelu ja reaalimaailman esteiden esiintyminen (Stallkamp et al. 2012). Liikennemerkit vaihtelevat myös alueellisesti, sillä esimerkiksi eläimistä varoittavat merkit voivat koskea vain tiettyjä alueita.

Kuvassa 1 esitetään otettujen panoraamakuvien sijainnit sekä Väyläviraston avoimen datan palvelusta saadut väistämisvelvollisuus liikenteessä -liikennemerkit Inkoon alueella. Väyläviraston datasta oli saatavilla vain valtion hallinnoimat kärkikolmiot. Kartalla ne on suodatettu käsittelemään vain Inkoon kuvausaluetta, jotta vertailu olisi mahdollista oman aineiston välillä. Lisäksi kuvassa on mukana esimerkki kyseisestä liikennemerkistä, johon viitataan työssä kärkikolmiona tekstin yksinkertaistamiseksi. Kuvassa 2 puolestaan esitetään Inkoon kuva-aineistosta havaittujen kärkikolmioiden kuvanottopaikat.




*Kuva 1. Väylän aineistoon merkityt kärkikolmiot ja esimerkkikuva kärkikolmiosta.*




*Kuva 2. Inkoon panoraamakuvilta manuaalisen tarkastelun pohjalta havaitut kärkikolmiot.*

## 2.2 Koneoppiminen

Koneoppimista voi pitää tekoälyn yhtenä osa-alueena, jossa kehitetään ja suunnitellaan algoritmeja sekä tekniikoita, joiden pohjalta kone oppii aineiston avulla (Lehto et al. 2019, Kanevski et al. 2009). Se eroaa perinteisestä ohjelmoinnista, jossa enemmänkin tuotetaan vastauksia aineiston ja jo ennalta määritettyjen sääntöjen perusteella. Koneoppimisessa pyritään löytämään nämä säännöt aineiston ja mahdollisesti saatavilla olevien vastausten avulla (Leppänen 2019, 19). Koneoppimisessa keskeistä on, että kone oppii aiempien kokemusten perusteella parantamaan tulevia suorituksia. Lehto et al. (2019) lisäävät, että tarkoituksena on saada ohjelmisto toimimaan paremmin pohja-aineiston sekä käyttäjän toiminnan perustella. Koneoppimista hyödyntäen ohjelma oppii tunnistamaan, luokittelemaan ja ennustamaan asioita.

Koneoppimista on onnistuttu hyödyntämään monin tavoin spatiaalisten aineistojen käsittelyssä ja analyyseissä. Esimerkiksi Pham et al. (2017) arvioivat koneoppimismenetelmin maanvyöryjen herkkyyttä Himalajalla pohjautuen esimerkiksi sääolosuhteisiin ja maanpinnan muotoihin sekä tyyppeihin. He onnistuivat kehittämään mallin, joka ennustaa todennäköisiä alueita maanvyöryille niiden herkkyyden perusteella. Muita tutkimuskohteita ovat olleet esimerkiksi eroosion mallinnus (Garosi et al. 2019), tulvariskien kartoitus (Mojaddadi et al. 2017) sekä Twitter-aineistoon perustuva influenssan leviämismallinnus (Allen et al. 2016).

Koneoppiminen jaetaan kahteen eri osaluokkaan: ohjattuihin ja ohjaamattomiin algoritmeihin (Kanevski et al. 2009, Leppänen 2019, Stenroos 2017). Joskus puhutaan myös puoliohjatuista ratkaisuista, joissa on osia molemmista edellä mainituista. Koneoppimiseen pohjautuvista konvoluutioneuroverkoista (*Convolutional Neural Networks = CNN*) puhutaan seuraavissa kappaleissa ja näiden koulutus toteutetaan yleensä ohjattuna oppimisena (Voulodimos 2017), kuten tyypillisesti muutkin koneoppimista hyödyntävät algoritmit (Stenroos 2017, 11).

## 2.3 Neuroverkot ja konvoluutioneuroverkot

### 2.3.1 Neuroverkkojen rakenne ja toiminta

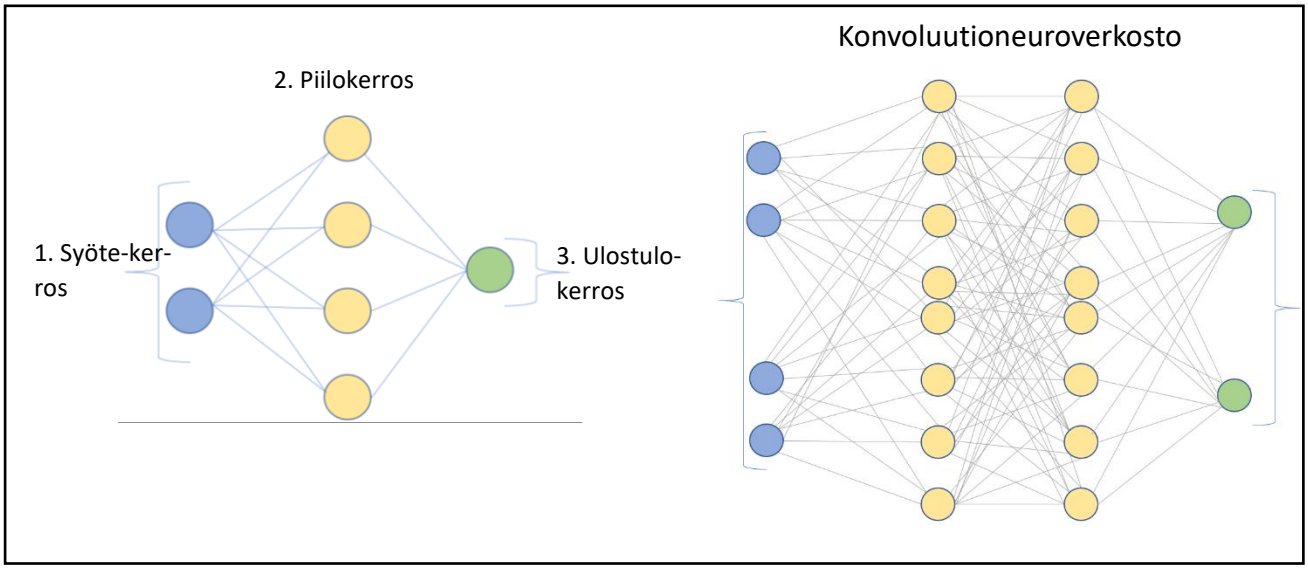
Keinotekoisten neuroverkkojen kehitys alkoi jo 1940-luvulla, mutta laskenta- ja tiedonhallintakapasiteetin nousun myötä niiden kouluttamiseen on tullut enemmän mahdollisuuksia 2010-luvulta alkaen (Lehto et al. 2019, kappale 2.5). Neuroverkot koostuvat yksiköistä eli neuroneista, joilla on verkostomainen yhteys muihin neuroneihin eri tasoilla. Neuroverkko mallintaa ihmisaivojen toimintaa,

mutta niitä ei voi lähtökohtaisesti kutsua samanlaiseksi kuin ihmisaivoja. Nämä ovat monin tavoin kompleksisemmat kuin keinotekoiset neuroverkot, jotka toimivat matemaattisina funktioina ja perustuvat yhdistävään laskentaan. (Lehto et al. 2019, Stenroos 2017).

Neuroverkkojen rakenteesta on tärkeää ymmärtää synapsien yhdistämät eri kerrokset, joita ovat:

1. Syötekerros (*input*), johon syötetään aineistoa
2. Piilokerros (*hidden layer*), joka käsittelee syötekerroksesta tulevan sisääntulon
3. Ulostulokerros (*output*), aineiston lopullinen muoto, joka riippuu piilokerroksen prosesseista.

Nämä kerrokset rakentuvat neuroneista, jotka koulutetaan erinäisin parametrein verkoston koulutuksen yhteydessä. Eri kerrosten neuronien määrä riippuu valittujen parametrien määrästä, esimerkiksi tietyt neuronit etsivät kohteentunnistuksessa tiettyä ominaisuutta, kuten väriä tai muotoa syötetystä aineistosta (Lehto et al. 2019). Kuvassa 3 esitetään yksinkertainen neuroverkko, jossa parametreja on vähän sekä monimutkaisempi syvä neuroverkko, joka koostuu useista syöte-, piilo- ja ulostulokerroksista.




*Kuva 3. Vasemmalla puolella kuva yksinkertaisesta neuroverkosta ja oikealla puolella monimutkaisempi konvoluutioneuroverkosto, mukautettu (Lehto et al 2019, kappale 4.1)*

### 2.3.2 Syväoppiminen

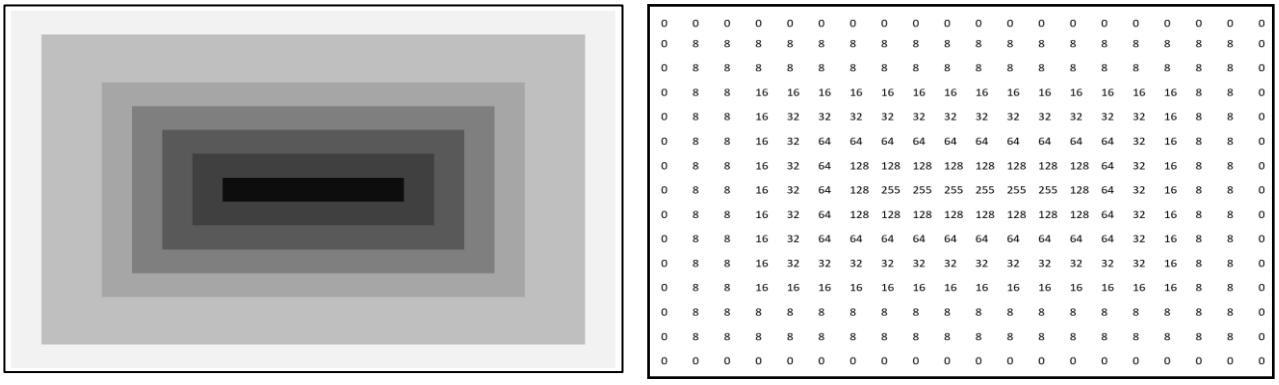
Syväoppiminen (*deep learning*) on yksi koneoppimisen osa-alueista ja käsitteellä viitataan konvoluutioneuroverkostoihin, joissa on enemmän kuin yksi piilokerros (Leppänen 2019). Piilokerroksille on määritetty omat tehtävänsä ja niitä voi olla verkostossa suuri määrä. Tämän myötä erilaisia valittavia parametreja on valtava määrä, joten verkoston kouluttaminen vaatii myös paljon aineistoa (Lehto et al. 2019). Näiden peräkkäin aseteltujen piilokerrosten neuronit ovat yhteydessä toisiinsa ja aineiston

liikkuessa neuroneissa algoritmi oppii merkityksellisiä esitysmuotoja aineistosta (Leppänen 2019). Syväoppiminen tarjoaa siis keinon laskentamallien moniprosessoiville tasoille oppia ja esittää aineistoa. Sen avulla voidaan esittää, miten aivot vastaanottavat ja ymmärtävät multimodaalia informaatiota (Ball et al. 2017, Voulodimos et al. 2017).

Syväoppimismenetelmät ovat olleet suosittuja konenäön sovellutuksissa ja ihmisnäön tapa käsitellä visuaalista informaatiota on ohjannut kohteentunnistamiseen käytettyjen konvoluutioneuroverkostojen rakenteen suunnittelua (Leppänen 2019, Voulodimos et al. 2017, 2). Syväoppimista on käytetty myös spatiaalisten aineistojen parissa esimerkiksi käsiteltäessä laajoja aineistoja, kuten ilmakuvia tai tarkkoja pistepilviaineistoja (Guo et al. 2020, 371). Syväoppimismenetelmien avulla on myös näiden lisäksi onnistuttu luokittelemaan satelliittikuvilta maanpeiteluokkia ja maankäyttöä kaupunkiympäristössä (Lv et al. 2014).

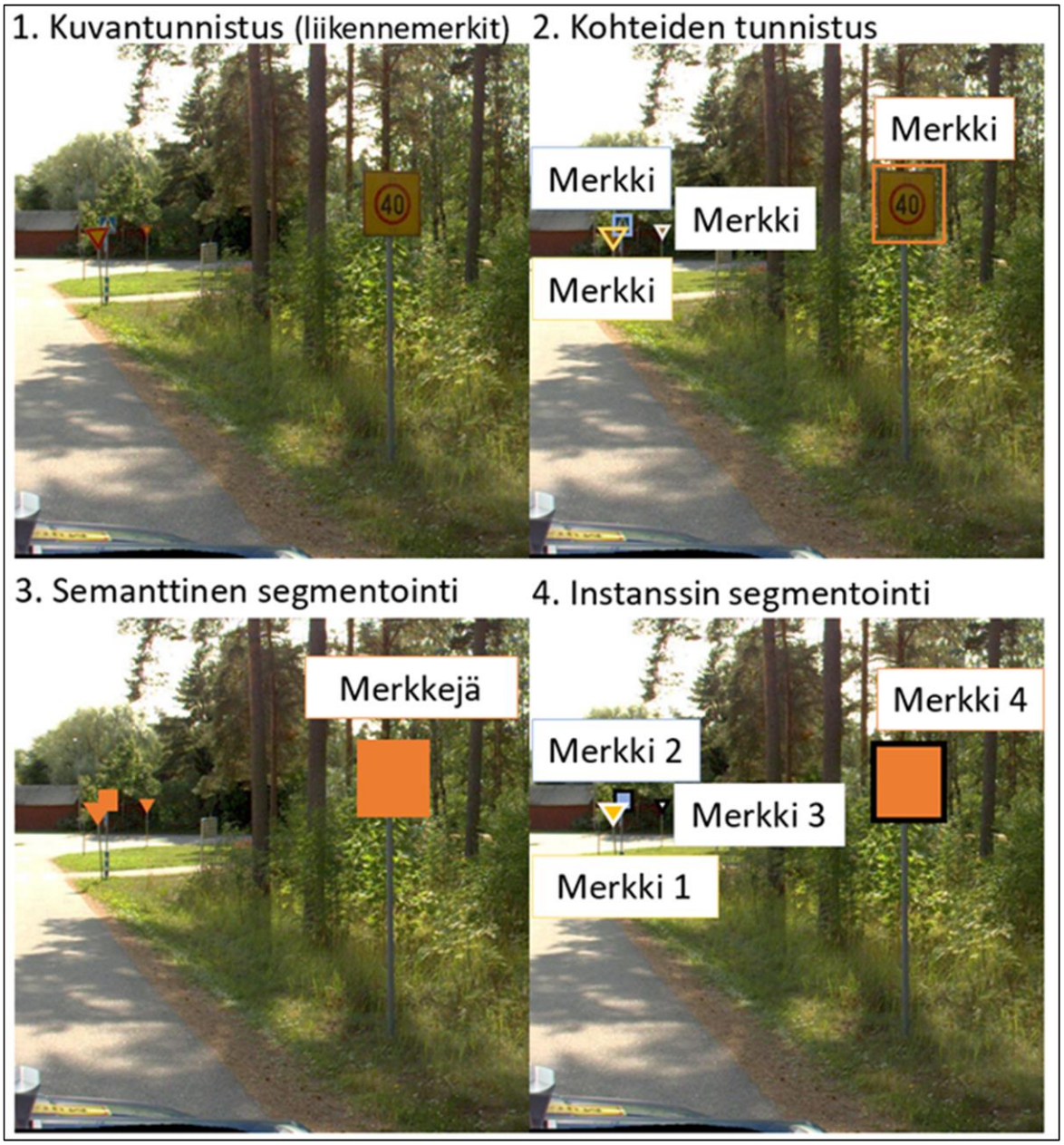
## 2.4 Konenäkö

Konenäöllä tarkoitetaan tietokoneavusteista digitaalisten kuvien tai videoiden prosessointia ja sen avulla saatavien piirteiden irrottamista (*extraction*) käsiteltävään muotoon (Stenroos 2017, Nixon & Aguado 2019). Konenäön avulla pyritään imitoimaan ihmisen kykyä nähdä asioita (Lehto et al. 2019) ja konenäköä hyödyntäen ohjelmia opetetaan havainnoimaan informaatiota samaan tapaan (Nixon & Aguado 2019). Kuvat koostuvat pisteistä tai kuvaelementeistä (kuten pikseleistä), joita tietokone analysoi ja järjestää ne numeroiksi tai numeroketjuiksi tietokoneelle (Nixon & Aguado 2019). Esimerkiksi kuvassa 4 esitetään, miten tietokone käsittelee harmaan väriskaalan kuvan. Vaaleat pikselit saavat arvon 0 ja sävyn tummentuessa keskikohtaa päin myös pikseliarvot kasvavat.




*Kuva 4. Miten tietokone näkee kuvan. Kuva pohjautuu Nixonin ja Aguadon (2019, 20) kuvaan.*

Ihmisnäkö on kehittynyt miljoonien vuosien saatossa ja se toimii visuaalisten ärsykkeiden avulla ja havainnoi näiden pohjalta informaatiota (Nixon & Aguado 2019, 3). Ihmisnäkö pystyy arvioimaan relatiivista etäisyyttä hyvin, mutta absoluuttinen etäisyyden mittaus onnistuu paremmin konenäön avulla (Nixon & Aguado 2019). Konenäön avulla pyritään kehittämään ratkaisuja niin kohteiden tunnistamiseen, kuvien luokitteluun, semanttiseen segmentointiin kuin instanssin segmentointiinkin (Wu et al. 2018). Tässä työssä tarkastellaan lähinnä kohteiden tunnistusta. Nämä eri ratkaisut on esitelty kuvassa 5. Kuvien luokittelussa kuvan kohteet assosioidaan tiettyyn luokkaan tai luokkiin, mutta niiden sijaintia tai rajautumista kuvalla ei tarkastella. Semanttisessa segmentaatiossa ennustetaan valitusta luokasta koostuvat kuvapikselit yhteen, mutta ei eroteta kohteita erillisinä, joka taas on mahdollista instanssi segmentaation avulla (Wu et al. 2019).




*Kuva 5. Konenäön eri sovellutuksia. Kuva mukailtu Wu et al. (2019, 1) kuvan mukaan.*

## 2.5 Kohteiden tunnistaminen

### 2.5.1 Kohteentunnistuksen tavoitteet

Kohteentunnistuksessa pyritään tunnistamaan tiettyjä luokiteltuja kohteita kuvilta (Voulodimos et al. 2017) tai videoilta (Hienonen 2014). Hienonen (2014) lisää paikantamisen olevan usein olennaisena osana tunnistuksessa. Tunnistukseen käytettävät algoritmit pyrkivät havaitsemaan mahdollisia kohteita kuvalta ja tämän perusteella luomaan joko suorakaiderajaukseen (*bounding box*) tai pikselimaskin kaltaisen tunnuksen tälle kohteelle (Wu et al. 2019). Kohteentunnistusta toteutetaan suurimmaksi osin koneoppimisen avulla (Stenroos 2017). Kohteentunnistuksessa algoritmeille pitää ensin syöttää suuri määrä kuva-aineistoa, jonka pohjalta algoritmit oppivat tunnistamaan samankaltaisia tai samankaltaisuuksia sisältäviä kokonaisuuksia kuvilta. Spatiaalisten aineistojen parissa kohteentunnistusta on käytetty tutkimuksen kohteena olevien liikennemerkkien inventoinnin lisäksi esimerkiksi kaukokartoituskuvilta tehtävään maanpeitteen tulkintaan (Li et al. 2020).

Mielenkiintoinen ongelma kohteentunnistuksessa on se, että sen onnistumiseksi pitää tunnistaa kohteen muoto (ja luokka), mutta myös sen sijainti kuvalla. Toisaalta sijainnin selvittäminen vaatii, että kohteen muoto on myös tunnistettu (Stenroos 2017, 18). Konvoluutioneuroverkkoja hyödyntävässä kohteentunnistuksessa ennustetut kohteet käydään verkostossa läpi. Tapa, jolla aineisto virtaa verkoston läpi riippuu valitusta verkostosta ja sen parametreista. Näiden valintojen perusteella verkoston eri osat arvioivat ennustettuja kohteita ja niiden luotettavuutta oikeana havaintona (Voulodomis et al. 2017).

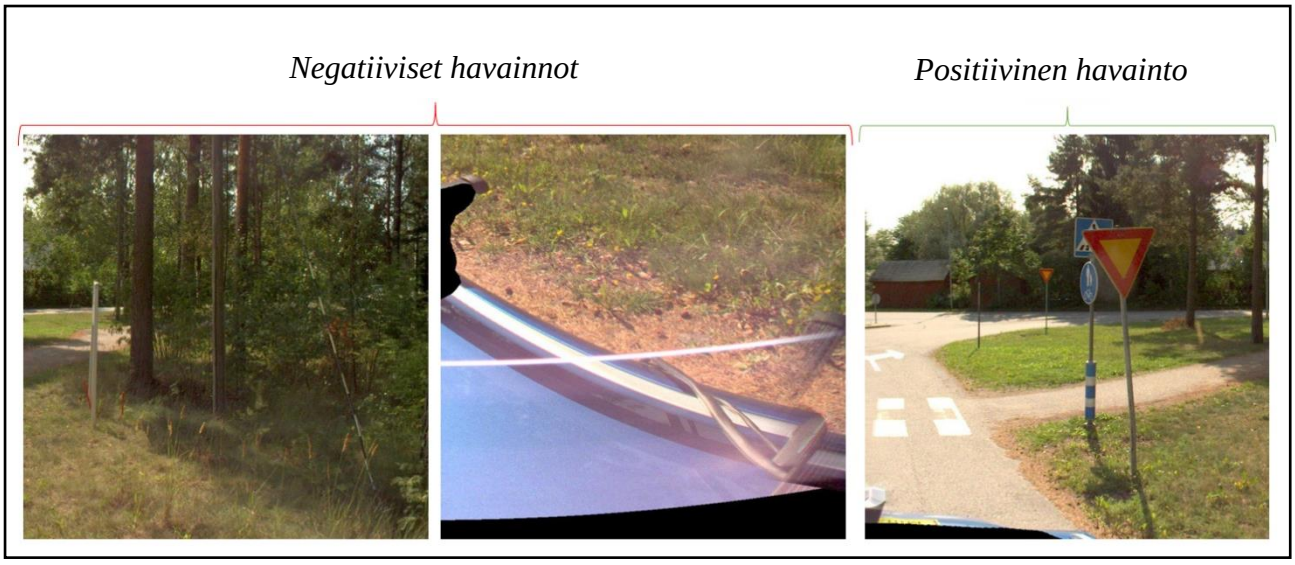
Kohteentunnistukseen kuuluu siis kolme eri aspektia (Du et al. 2018, Hienonen 2014, Zhu et al. 2016):

1. havaitseminen
2. paikallistaminen
3. luokittelu.

Luokittelussa kuvalta pyritään siis tulkitsemaan kohteen luokka tai tyyppi, paikallistamisessa kohteen sijainti kuvassa sekä kohteiden havaitsemissa kohteen olemassaolo kuvalla.

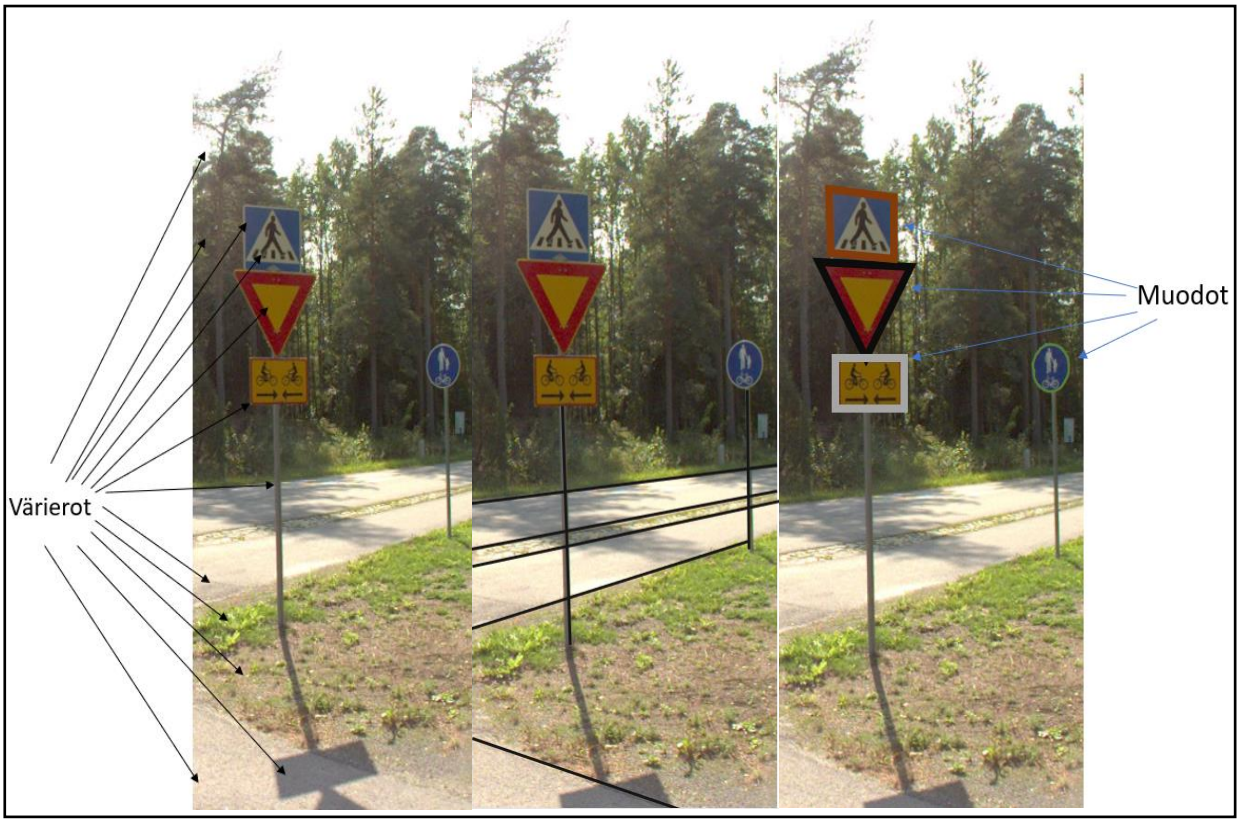
### 2.5.2 Kohteentunnistamisen koulutus ja testaus

Kohteentunnistusneuroverkon koulutukseen käytetään positiivisia havaintoja halutusta kohteesta (kuva 6). Koulutusaineistossa on oltava riittävä määrä näitä, jotta voidaan varmistua algoritmin tunnistuskyvystä myös vaikeille kohteille. Koulutuksessa haluttu kohde rajataan kuvalta esimerkiksi *bounding box* -menetelmällä. Tällöin osa rajauksesta sisältää myös elementtejä, jotka eivät ole kohteiden (tässä tapauksessa kärkikolmion) tunnistamiseen relevantteja. Tämän vuoksi taustan suositaan olevan mahdollisimman monipuolista, sillä liikennemerkit voivat sijaita erilaisissa ympäristöissä sisältäen runsaasti erilaisia taustoja. Rajausten laadun todetaankin olevan erittäin tärkeää kohteentunnistusneuroverkon lopullisen toiminnan varmistamisessa (Li et al. 2019). Koulutusaineistoa löytyy esimerkiksi liikennemerkkien osalta monesta eri lähteestä ja sitä on mahdollista kerätä myös omatoimisesti, kuten tässä työssä.



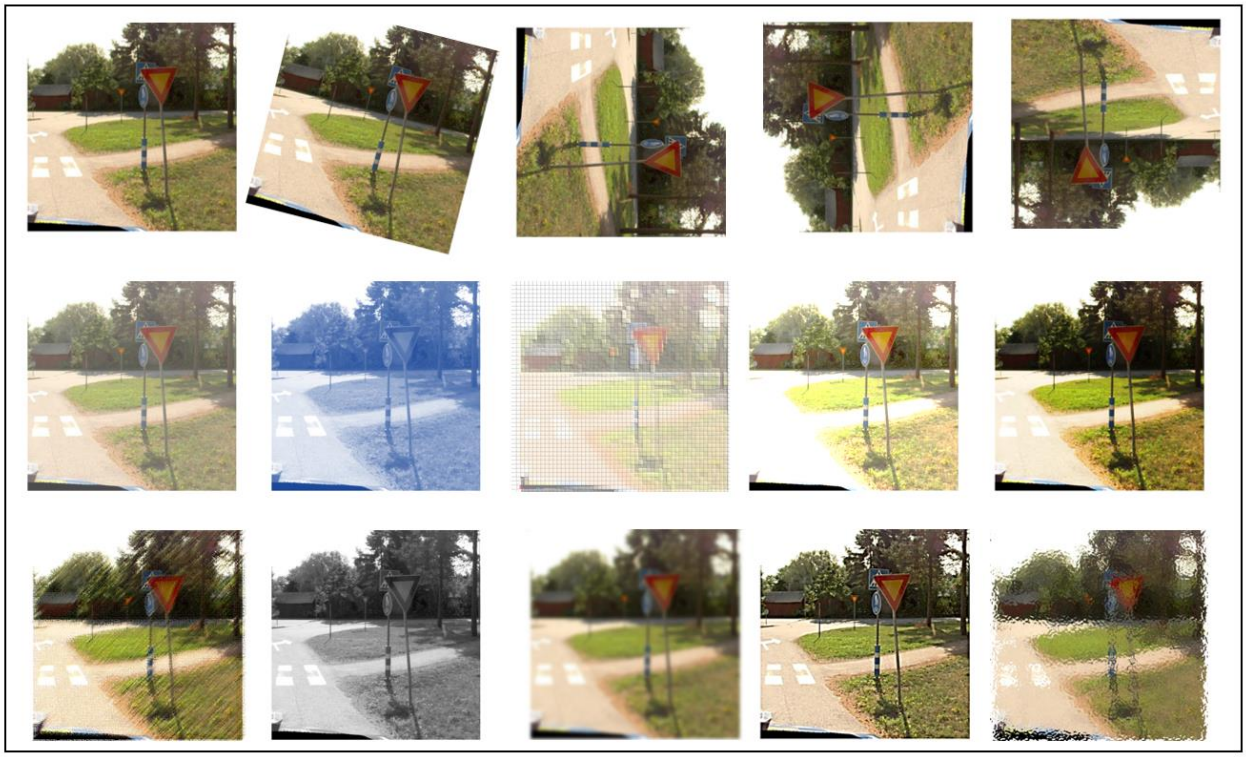

*Kuva 6. Kuvien läpikäynnin yhteydessä havaitaan positiiviset havainnot, joita käyttää koulutusaineistona. Negatiivisia havaintoja ei käytetä koulutuksessa, mutta testauksessa niiden roolina on toimia algoritmin validointiin. Mikäli algoritmit tunnistavat havaintoja kuvilta, joissa niitä ei ole, on kyseessä väärä positiivinen havainto (false positive).*

Kuvassa 7 näkyy, minkälaisia piirteitä algoritmi oppii tunnistamaan kuvilta sekä millaisia nämä ovat. Testauksessa verkosto pyrkii tunnistamaan koulutusaineiston pohjalta näitä piirteitä ja tekee niiden myötä ennustuksia. Nämä piirteet vaihtelevat tunnistettavan kohteen mukaan. Esimerkiksi liikennemerkkien kohdalla osa merkeistä muistuttaa toisiaan, mikä voi aiheuttaa haasteita tunnistuksen toimintaan ja aiheuttaa vääriä positiivisia havaintoja.



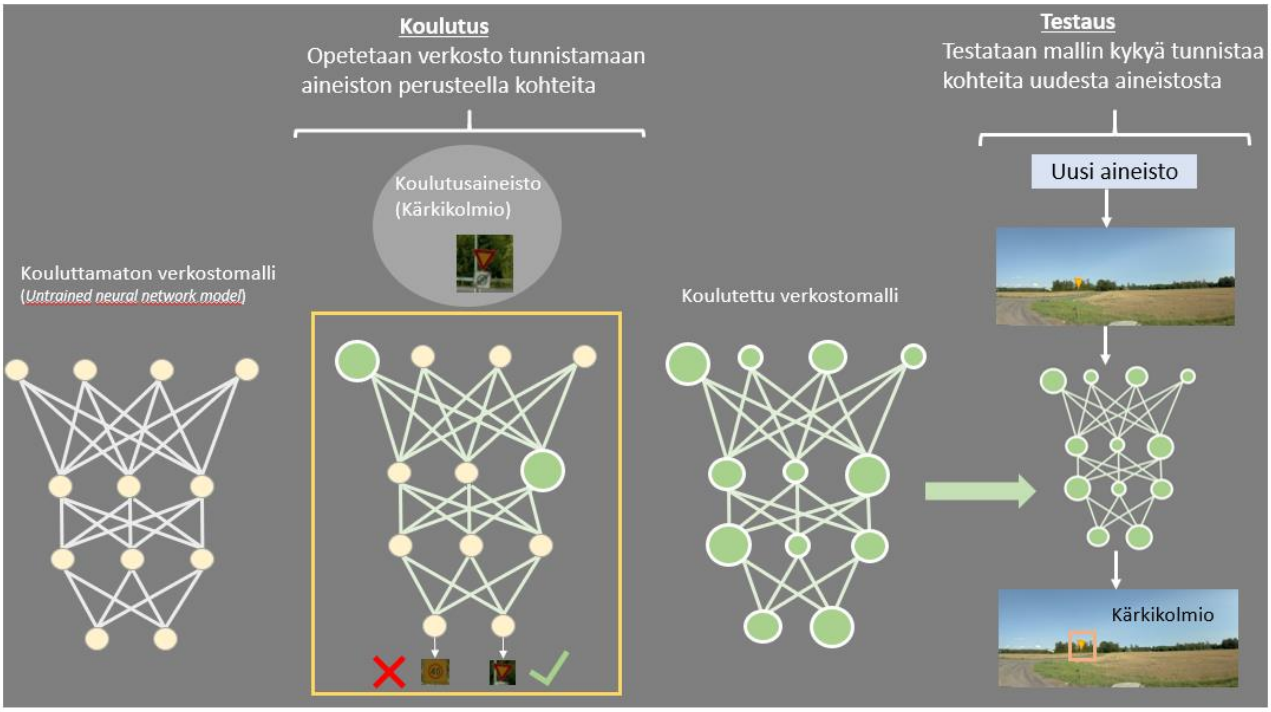

*Kuva 7. Kohteentunnistusalgoritmi oppii tunnistamaan erinäisiä piirteitä kuvilta, kuten värieroja, rajautumisia sekä muotoja.*

Kohteentunnistamisessa on tärkeää pyrkiä välttämään ylisovittamista (*overfitting*), eli tilannetta, jossa tehty malli sopii liian hyvin aineistoon. Tätä voidaan välttää erinäisiä apuvälineitä hyödyntämällä, mikä samalla parantaa koulutuksen laatua (Voulodimos 2017, 4). Näitä apuvälineitä ovat muun muassa aineiston augmentointi, epätasapainotettu otanta (*imbalanced sampling*), kaskaadioppiminen (*cascade learning*) sekä paikallistamisen jalostus (*localization refinement*) (Wu et al. 2019). Esimerkiksi augmentoinnissa muunnellaan alkuperäisen kuvan visuaalista ilmettä väritasapainoja, muotoja ja muita ominaisuuksia muuttaen (Wu et al. 2019). Kuvassa 8 näkyy esimerkkejä erilaisista augmentointimetodeista. Tämän myötä yhdestä kuvasta muodostuu uusia kuvia, joiden avulla uutta koulutusaineistoa saadaan lisää.



*Kuva 8. Esimerkkejä augmentoiduista kuvista.*

Koulutukseen tuodaan kouluttamaton verkosto, joka koulutetaan näkemään kärkikolmioita ja tunnistamaan niitä. Koulutusosiossa voidaan käyttää aiemmin mainittuja apuvälineitä, kuten augmentointimenetelmiä koulutuksen laadun parantamiseksi. Verkoston eri osat oppivat tunnistamaan tiettyjä piirteitä kuvilta, sekä arvioivat niiden oikeellisuuden todennäköisyyttä vertaamalla näitä toisten verkoston osien tunnistamiin piirteisiin. Testauksessa taas verkoston eri osat arvioivat kuvan eri kohdissa olevan piirteitä, jotka viittaavat tunnistettavaan kohteeseen. Mikäli näitä on useampia kuvan jollain alueella, luokitellaan se todennäköiseksi havainnoksi ja sille annetaan luotettavuusarvio. Tätä prosessia havainnollistetaan kuvassa 9.




*Kuva 9. Testauksessa testataan koulutetun neuroverkon kykyä tunnistaa uudesta aineistosta kohteita. Mukailtu Nvidian (2016) artikkelin pohjalta.*

## 2.6 Konvoluutioneuroverkostojen esittely

### 2.6.1 Konvoluutioneuroverkostojen rakenne

Konvoluutioneuroverkot (CNN) ovat neuraalisten verkkojen muunnelma, jota voi hyödyntää varsinkin kohteiden tunnistuksessa, paikallistamisessa ja luokittelussa (Zhu et al. 2016). Niiden rakenteen suunnittelua on ohjannut ihmisnäön tapa käsitellä visuaalista informaatiota (Voulodimos 2017, 2). Konvoluutioneuroverkot (CNN) tarjoavat tehokkaan matemaattisen tavan ilmentää ja irrottaa (*extract*) piirteitä siihen syötetystä aineistosta. CNN pystyy myös tunnistamaan jo havaitsemansa kohteet ja näin nopeuttamaan verkoston toimintaa (Du 2018).

Alun perin konvoluutionverkostojen kohteentunnistusmahdollisuudet rajoittuivat yksittäisten kohteiden havaitsemiseen kuvilta. Useampien kohteiden samanaikainen havainnointi muuttui mahdolliseksi R-CNN (*Region based Convolutional Neural Networks*) -pohjaisten algoritmien kehityksen myötä (Du 2018). R-CNN tunnistaa kuvilta kohteita jakamalla kuvan eri osiin. Jakaminen kuitenkin hidastaa tunnistusta ja vaatii myös itsessään paljon laskentatehoa.

Seuraavaksi kehitettiin Fast R-CNN, jonka avulla prosessointi tuotettiin yhdessä putkessa (*pipeline)* verrattuna R-CNN:n monimutkaisempaan rakenteeseen. Koulutuksessa hyödynnetään lisäksi ROI (*Regions of Interest*) -yhdistämistä (*pooling*), joka nopeuttaa prosessointia. Viimeinen virstanpylväs

oli Faster R-CNN, joka hyödyntää edellä mainittujen lisäksi RPN (*Region Proposal Networks*) -verkostoja jakaakseen konvoluutiotasoja samanaikaisesti kohteentunnistusverkostotasojen kanssa.

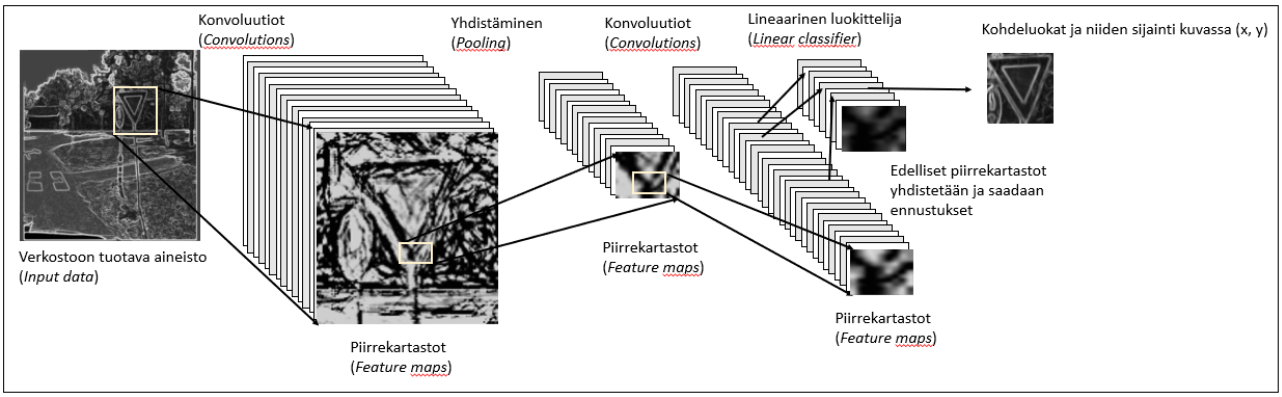
Seuraavaksi käydään läpi tarkemmin konvoluutioneuroverkoston rakennetta sekä mitä sen eri osat tekevät verkostossa. Eri osien selitykset on koostettu useammasta lähteestä, joissa ne ovat lähes yhteneväiset (esim. Voulodimos et al. 2017, Du et al. 2018, Liu et al. 2019). Konvoluutioneuroverkoston arkkitehtuuria kuvataan usein putkena (*pipeline*), jonka alkupäässä on lähtöaineisto, sen jälkeen on eri parametrein säädettyjä tasoja sekä lopussa tunnistustaso. Näiden läpi aineisto virtaa putkessa ja näin saadaan ennustetut tunnistukset.

Voulodimos et al. (2018) kertovat, että CNN-arkkitehtuuri koostuu kolmesta eri konkreettisesta ajatuksesta:

1. Paikalliset vastaanottavat kentät (*local receptive fields*)
2. Tasatut painot (*tied weights*)
3. Osanäytteistäminen (*Subsampling*)

Ensimmäiseen pohjautuen konvoluutioverkon eri osat saavat syötteen niihin yhdistyvistä osista. Tämän myötä neuronit voivat irrottaa (*extract*) visuaalisia piirteitä kuten kulmia tai rajoja lähdeaineistosta. Tämän jälkeen näitä yhdistellään, minkä pohjalta voidaan luoda suurempia kokonaisuuksia (Voulodimos et al. 2017). Tasatuilla painoilla viitataan yksinkertaisten piirretunnistimien (*feature detectors*) käyttöön koko kuvalla, jos ne ovat onnistuneet tunnistamaan piirteitä kuvan tietystä alueesta. Konseptissa ryhmänosien ajatellaan sisältävän identtiset painoarvot ja nämä ryhmänosat ovat neuroverkossa organisoitu tasaisesti pinnalle (*planes*). Nämä ryhmänosat ovat erillään muista ryhmistä ja ne tunnistavat tiettyjä piirteitä pinnasta riippuen. Näiden pohjalta syntyy piirreverkosto (*feature map*), joka muodostetaan monella eri konvoluutiotasolla.

Kuvan 10 visualisoinnin tarkoituksena on havainnollistaa, miltä verkoston toiminta voisi näyttää. Ensin aineisto tuodaan verkoston sisään, siitä tunnistetaan piirrekartastoja eli osia mahdollisista kohteista. Näitä yhdistetään ja käsitellään uudestaan konvoluutioissa, minkä pohjalta tehdään ennusteet havaituista kohdeluokista ja niiden sijainnista kuvassa.




*Kuva 10. CNN-arkkitehtuuri kuvattuna kohteen tunnistamistehtävässä. Kuva mukailtu Voulodimos et al. kuvan pohjalta (2017, kuva 1).*

Konvoluutioverkostojen rakenteesta on hyvä ymmärtää, että sen osat voivat myös toistua putkessa ja niitä voidaan käyttää myös yhdessä samanaikaisesti. Eri konvoluutioverkoston osat ja niiden toiminta esitetään tarkemmin seuraavaksi. Sisääntulokerroksen (*Input layer)* tehtävänä on muuntaa lähtöaineistona toimivan kuvan mittasuhteet samalle skaalalle nollakeskuisiksi (*zero-centered*), minkä jälkeen skaala normalisoidaan. Tarkoituksena on siis muuntaa aineisto muotoon, jossa käsittely olisi nopeaa (Du 2018). Aineisto käsitellään myös *Principal Components Analysis* (PCA) avusteisesti, jotta aineiston ulottuvuudet koskettaisivat muutamaa päätekijää ja täten käsittely helpottuisi entisestään.

Konvoluutiotaso (*Convolutional = CONV*) muodostaa CNN ytimen, jolla filtteröidään CONV-tason avulla alkuperäistä kuvaa, täten luoden alueita yksittäisistä pikseleistä tai näiden osajoukoista (Du 2018, Voulodimos et al. 2017). Näistä CONV tunnistaa mallikuvioita (*pattern*) ja jokaisessa verkoston vaiheessa aineistoa filtteröidään ja valitaan mallikuvioita määritellyllä tavalla. Jotkut filtterit voivat esimerkiksi havainnoida vain ympyränmuotoisia ja toiset neliönmuotoisia kohteita. CONV-taso myös päättelee, onko soluista löytyvä piirre osa yhtä yhtenäistä kohdetta vai jotain muuta. Lopulta koostuu piirreverkosto (*feature map*), joka voidaan luokitella ennustuksena (*prediction*) yksittäisestä kohteesta.

CONV-tason tarkkuus riippuu valituista parametreista. Esimerkiksi *Stride*-arvon perusteella päätetään, kuinka ison osan pikseleistä filtteri käsittelee kerrallaan. Mitä suurempi arvo, sitä nopeammin filtteri käsittelee pikseleitä kerrallaan, mutta sitä epätarkempi lopputulos saadaan. Toisaalta liian pienellä *Stride*-arvolla laskentatehoa vaaditaan enemmän ja käsittely on hidasta. Tietokone käsittelee kuvia numerosarjoina ja kuvan tullessa CONV-tasoon se yrittää nähdä siinä piirteitä, jotka on myös aiemmin havaittu koulutusaineistossa. Esimerkiksi liikennemerkin kaari esitetään nollaa suurempina

ja muu tausta saa nolla-arvon. Mitä enemmän numeroita sisältäviä pikseleitä kone havaitsee, sitä todennäköisempi potentiaalinen tunnistus on.

Aktiivinen taso (*Active layer*) muokkaa CONV-tason antamat tulokset tai ennusteet nonlineaariseksi, koska aiemmin aineiston gradientti on osiltaan hävinnyt alisovittamisen (*underfitting*) takia (Du 2018). Yhdistämistaso (*Pooling layer)* vähentää CONV-tason tuloksien spatiaalisia ulottuvuuksia, kuten korkeutta ja leveyttä. (Du 2018, Voulodimos 2017). Yhdistämistä ja sovittamista on kolmen tyyppistä. Näitä ovat *General pooling*, *overlapping pooling* ja *spatial pyramid pooling* (SPP). Esimerkiksi SPP muokkaa erikokoisten kuvien ominaisuuksia siten, että niillä on sama määrä ulottuvuuksia. Tämän myötä esimerkiksi tietoa ei menetetä kuvien rajauksen myötä. Voulodimos et al. (2017) kuitenkin toteavat, että yhdistämisoperaatioita voidaan myös kutsua termeillä *downsampling* tai *subsampling*. Näillä viitataan informaation menettämiseen, mikä johtuu kuvan koon pienentämisestä. Voulodimos et al. (2017) toteavat tämän olevan lopulta hyödyllistä, sillä laskentatehoa saadaan lisää ja ylisovittamisen riski pienenee. Du (2018) ja Voulodimos et al. (2017) jatkavat yleisen yhdistämisen tavallisiksi strategioiksi keskiarvon tai maksimin mukaisen yhdistämisen.

Kokonaan yhdistetty taso (*Fully connected layer*) on usein viimeisenä konvoluutioverkostossa. Niiden funktiona on toimittaa aineisto ulostulotasoon (*output layer*) sekä yksinkertaistaa ja nopeuttaa aineiston laskentaa (Du 2018). Voulodimos et al. (2017) mukaan *fully connected*-taso muuntaa kaksiulotteiset piirrejoukot (*feature map*) yksiulotteisiksi kohdevektoreiksi (*feature vector*). Tämä kohdevektori taas voidaan luokitella tiettyyn luokkaan tai jatkokäsitellä se osana isompaa kokonaisuutta. Jälkimmäisenä mainittu tarkoittaa, että havaittu kohde ei ole vielä luokiteltavissa, mutta se voi olla osa isompaa kohdeosajoukkoa.

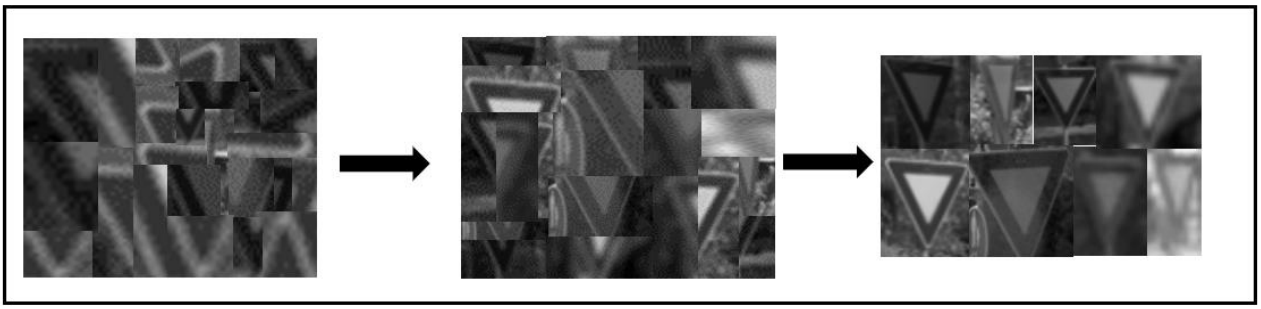
Du (2018) jatkaa, että muitakin tasoja voi olla, kuten regressiotasot tai *dropout*-taso, jonka avulla voidaan ratkaista ylisovittamista. Regressiotasoilla voidaan arvioida todennäköisyyksiä eri kohdeluokille. Lisäksi verkoston testauksen yhteydessä voi tapahtua päällekkäisiä havaintoja, eli duplikaattitunnistuksia samasta kohteesta. Tämä hidastaa algoritmien toimintaa sekä huonontaa tunnistuksen tasoa (Wu et al. 2019). Tätä varten verkostossa käytetään usein NMS (*Non-Maximum Supression*) -tasoa apuna. Tällä tasolla käsitellään päällekkäin osuvia tai duplikaattitunnistuksia ja pyritään poistamaan muut kuin luotettavimmat tunnistukset (Stenroos 2017, Wu 2019).

### 2.6.2 Algoritmien toiminta kohteiden tunnistamisessa

Kohteentunnistukseen käytettävät algoritmit jaotellaan yleensä kahteen luokkaan niiden rakenteen mukaan: yhden vaiheen tunnistukseen (*one-stage detection framework*) sekä kahden vaiheen tunnistukseen (*two-stage detection framework*) (Li et al. 2019). Suurin ero näiden välillä on, että kahden vaiheen tunnistuksessa lähdeaineistosta tehdään ensin ennusteet (*proposals*) mahdollisista kohteista, minkä jälkeen konvoluutioneuroverkko luokittelee nämä kohteet (Li et al. 2019, Wu et al. 2019).

Luokittelun pohjalta syntyy piirrevektorit (*feature vector*), joissa näkyy esimerkiksi pieniä osia yhdestä suuremmasta kohteesta. Esimerkiksi neliönmuotoisen liikennemerkin yksi sivu tai, kohteen ollessa pieni, mahdollisesti koko merkin neliö voi erottautua. Yhden vaiheen tunnistuksessa nämä tehdään samanaikaisesti (Li et al. 2019, Wu. et al. 2019) eikä niillä ole erillistä ennusteosiota. Tällöin piirteiden irrotus (*extraction*), kandidaattikohteiden suorakulmarajausregressio (*bounding box regression*) sekä luokittelu tehdään samanaikaisesti.

Yhden vaiheen tunnistuksen algoritmit myös arvioivat kaikkien kuvaosien sisältävän potentiaalisesti kohteen ja yrittävät luokitella osan joko kohteeksi tai taka-alaksi (Wu et al. 2019). Kuvassa 11 nähdään alhaalta ylöspäin, miten verkoston ensimmäiset tasot ennustavat yksinkertaisempia piirrevektoreita kuten kulmia ja reunoja. Näitä verrataan toisiin samalta alueelta ennustettuihin piirrevektoreihin, joiden pohjalta yleistyy piirreverkosto ja tämän myötä yleinen ennuste havaitusta kohteesta.



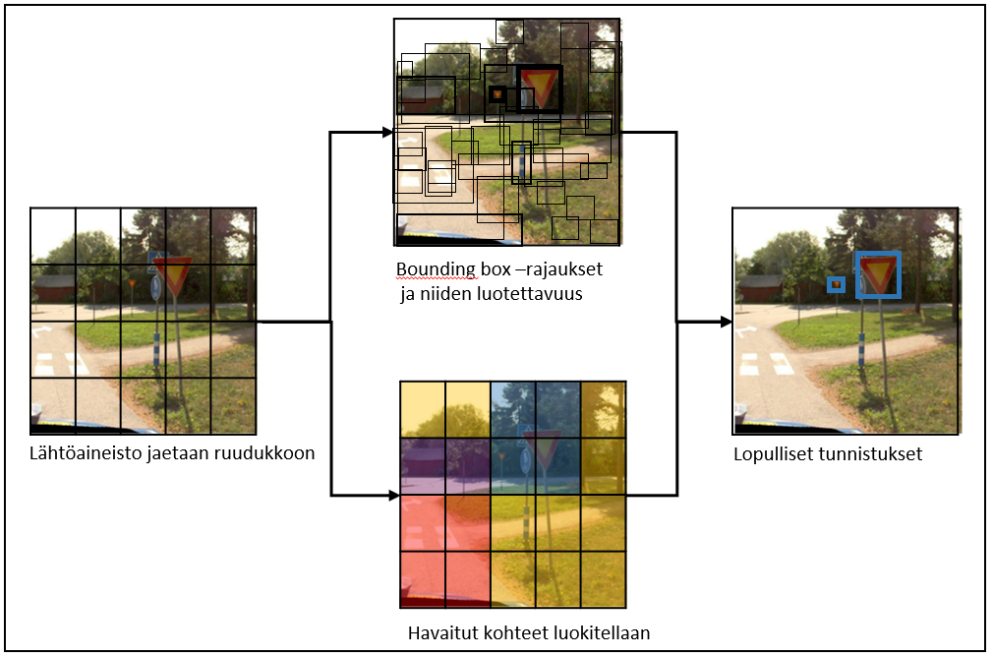
*Kuva 11. Miltä piirteiden irrotus näyttää eri konvoluutioverkoston tasoilla. Kuva mukautettu Matthew (2019) artikkelista.*

### 2.6.3 Yhden vaiheen tunnistus - YOLO

Yhden vaiheen tunnistus YOLO (*You Only Look Once*) on kohteentunnistusviitekehys (*object detection framework*), jota käytetään tässä työssä kohteentunnistukseen. YOLO-verkoston lähtökohtana on käsitellä kohteentunnistusta regressio-ongelmana (Redmon et al. 2016, Wu et al. 2019). Tällä tar-

koitetaan tulosteen (*output*) olevan numeerinen eli jatkuva, kun taas luokittelua hyödyntävät koneoppimistekniikat tuottavat kategorisen eli diskreetin tulosteen (Du 2018). YOLO:n konvoluutioneuroverkko on yksisuuntainen jatkumo, mikä ennustaa yhtäaikaisesti *bounding box* -rajaukset sekä luokittelutodennäköisyydet. YOLO:ssa siis ennustetaan yhdellä kertaa mitkä kohteet näkyvät kuvassa ja missä ne siinä sijaitsevat (Redmon et al. 2016). YOLO:n eri versioissa on eri määrä konvoluutiotasoja, esimerkiksi YOLOv1:ssa on 24 ja YOLOv3:ssa on 106 konvoluutiotasoa.

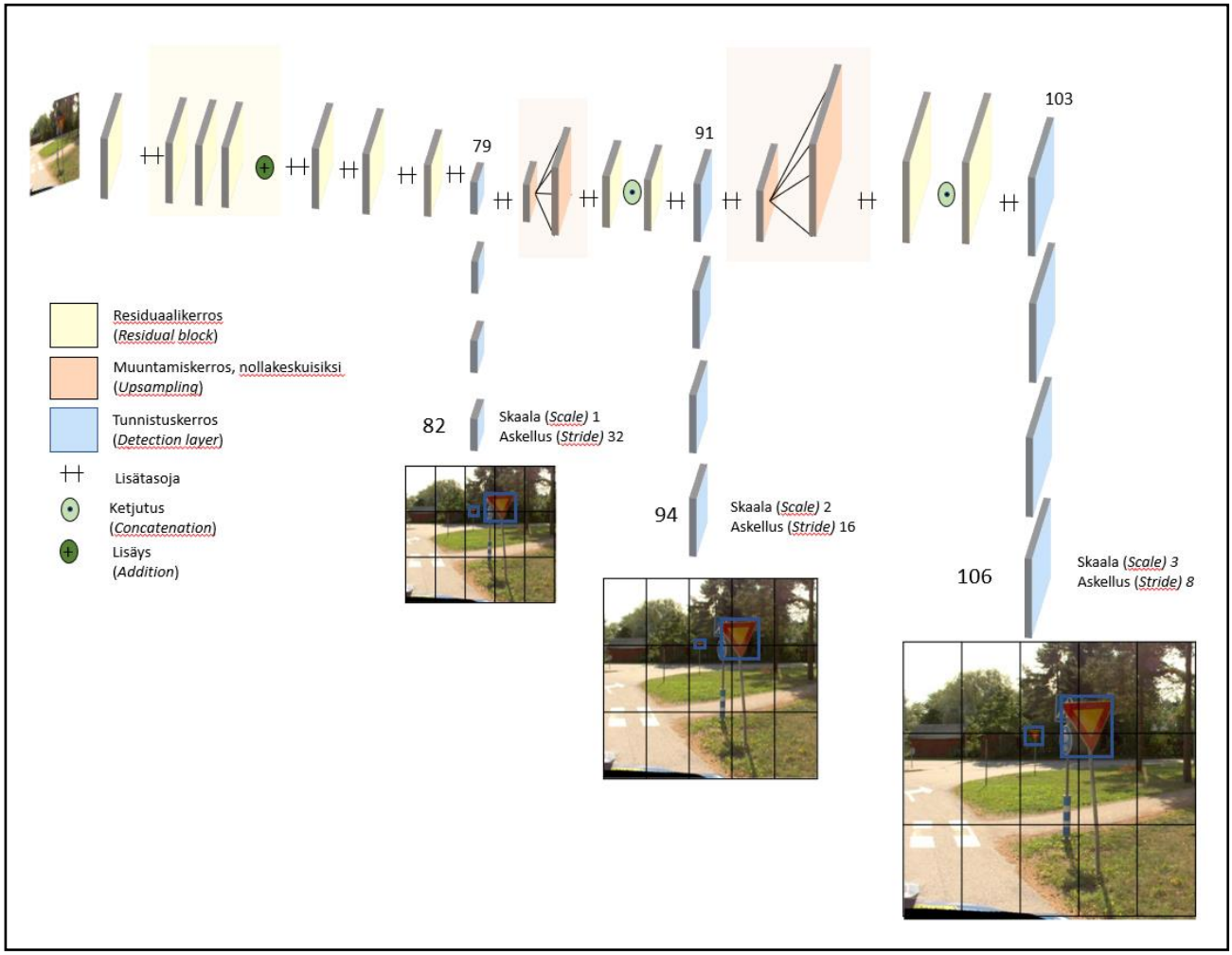
Kuvassa 12 nähdään miten YOLOv1 luo testauksessa ensin ruudukon kuvan päälle ja ruudukot jaetaan luokkatodennäköisyyksien mukaan omiin luokkiinsa. Yhdestä ruudusta muodostetaan *bounding box* -rajaus kohteille perustuen niiden luotettavuuteen oikeana havaintona. Lisäksi tämä kohdehavainto luokitellaan. Samalla ennustetaan kuvan kohteiden sijainti *bounding box* -rajauksin. Tämän jälkeen nämä yhdistetään ja saadaan lopullinen kohteentunnistustulos (Redmon et al. 2016).




*Kuva 12. YOLOv1:n toimintamalli kuvattuna. Kuva mukailtu Redmon et al. (2016, kuva 2) artikkelista.*

Kuvassa 13 puolestaan näytetään, miltä YOLO:n kolmannessa versiossa eri tunnistustasot vaihtelevat siten, että samasta kuvasta luodaan kolme tunnistusta. Ensimmäinen tunnistustaso (82) havainnoi suuria kohteita, toinen (94) keskisuuria kohteita sekä kolmas taso (106) pieniä kohteita. Tasojen välissä aineistoa muunnetaan erilaisten parametrien mukaisesti. Esimerkiksi aiemmin ennustettua piirrekartastoa (*feature map*) yhdistetään ketjutuksessa (*concatenation)* nollakeskuisiksi muunnettuihin kohdehavaintoihin (Redmon et al. 2018). Residuaalikerroksissa aineisto virtaa seuraavan tason lisäksi

useampiin tasoihin auttaen täten koulutuksessa, kun havaintoja voidaan tehdä samanaikaisesti monella tasolla. Samalla aineiston pituuskaltevuus (*gradient*) säilyy (Ju et al. 2019, Khan et al. 2019).

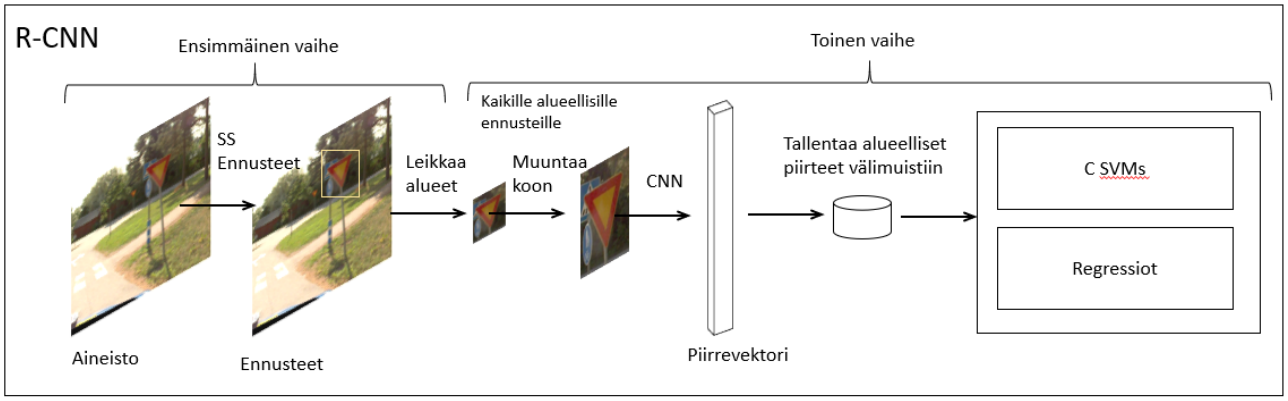



*Kuva 13. YOLOv3:n arkkitehtuuri. Kuva mukailtu Kathurian (2020) artikkelista.*

### 2.6.4 R-CNN-pohjaiset ratkaisut

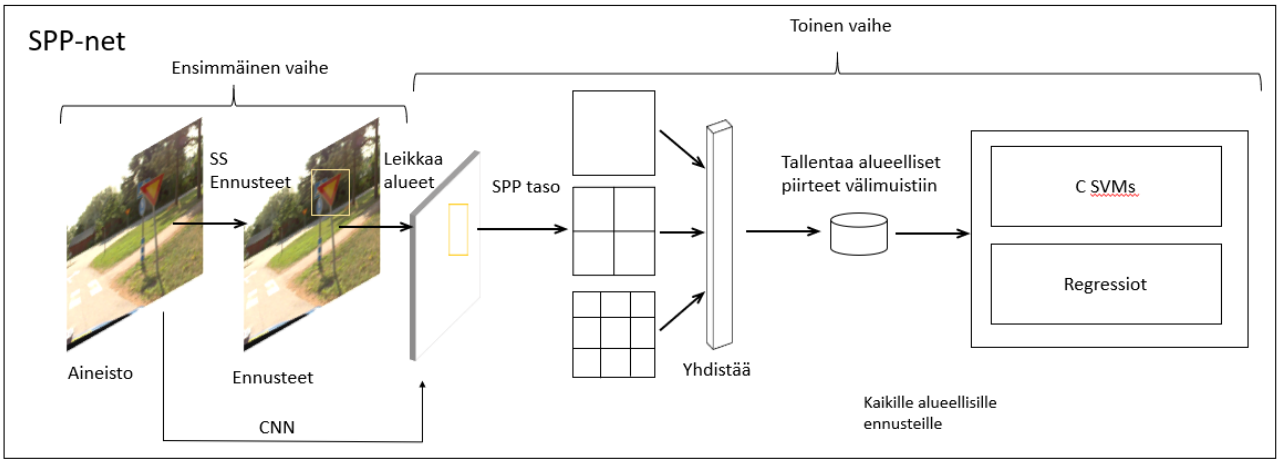
Seuraavaksi esiteltävät algoritmit pohjautuvat kaksivaiheeseen tunnistamiseen. Nämä pohjautuvat aluepohjaisiin ennustuksiin (*region based proposal*) sekä näiden perusteella tehtäviin luokitteluihin (Liu et al. 2019, 278). Tunnistus tapahtuu siis kahdessa vaiheessa toisin kuin YOLO:ssa, jossa nämä tehdään samanaikaisesti. Pyrkimyksenä on ennustaa alueet, joissa on mahdollisimman korkea *recall* -arvo, jotta kuvan kaikki kohteet luokitellaan edes johonkin näistä alueellisista ennustuksista. Myös tausta-alueet, jotka eivät kuulu mihinkään kohdeluokkaan, luokitellaan kahden vaiheen tunnistuksessa (Wu et al. 2019, 9). Muitakin kuin alla mainittuja tunnistusalgoritmeja on olemassa, mutta R-CNN ja sen eri mukaelmat (Fast R-CNN, Faster R-CNN) sekä SPP-net mainitaan useissa artikkeleissa (Du 2018, Liu et al., 2019, Voulodimos et al. 2017, Wu et al. 2019).

R-CNN (kuva 14) tarjosi ensimmäisenä keinon hyödyntää neuroverkon kykyä eristää ja luokitella piirteitä kuvan alueilta, minkä perusteella kuvilta tunnistetaan kohteita. Se rajaa (*crop*) ja vääntää (*wrap*) ehdotetut alueet normalisoidakseen ne ja standardisoidakseen niiden koot (Du 2018). Ensin se ennustaa potentiaaliset kohteet *bounding box* -rajauksilla kuvalta ja sen jälkeen neuroverkoston luokittelijaosa käy läpi nämä rajaukset. Jälkiprosessoinnissa nämä rajaukset käsitellään tarkemmiksi, duplikaatit poistetaan sekä näiden rajauksien tarkkuus pohjautuen muihin rajauksiin arvioidaan (Redmon et al. 2016). Ongelmana on laskentakapasiteetin määrä sekä rajauksen ja vääntämisen aiheuttaman informaation katoaminen (Du 2018). Lisäksi verkoston koostuessa monimutkaisista osista, on sen optimointi haastavaa, koska kaikki verkoston osat pitää kouluttaa erikseen (Redmon et al. 2016, Wu et al. 2019).



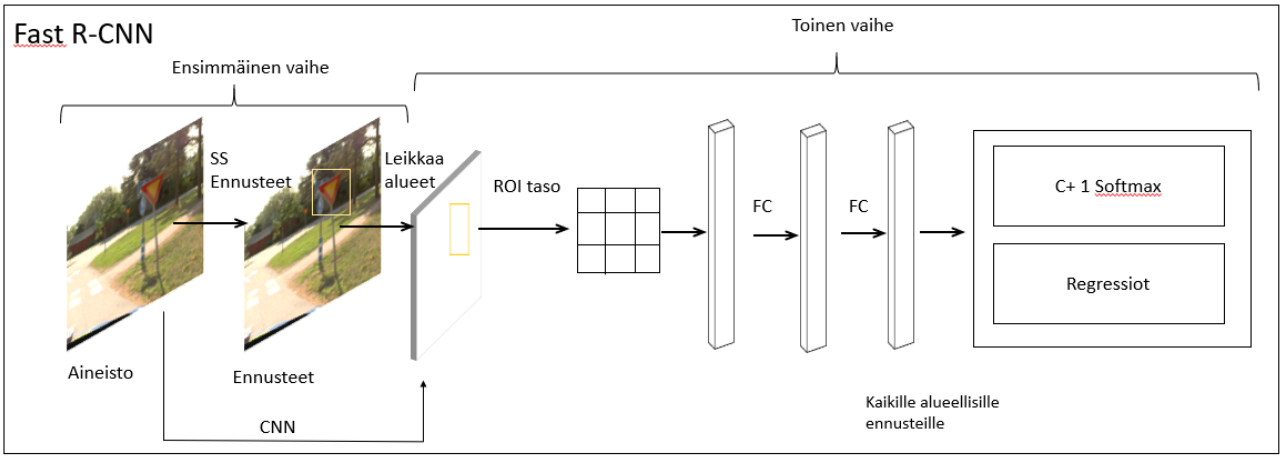

*Kuva 14. Kuva R-CNN arkkitehtuurista, kuva mukailtu Wu et al. (2019, kuva 4) pohjalta.*

SPP-net (*Spatial Pyramid Pooling Network)* kehitettiin R-CNN:n tunnistustarkkuuden parantamiseksi ja varsinkin sen eri kohteiden erottelukyvyn kehittämiseksi. SPP-net ei erikseen rajaa ehdokaskohteen sisältämiä alueita yksitellen konvoluutioneuroverkoston läpi, vaan se käsittelee koko kuvan kerrallaan (kuva 15, CNN-viiva sisääntulon ja SPP-tason välillä). Se ei myöskään muunna kuvan kokoa kuten R-CNN, joten tietoa ei menetetä tässä prosessissa (Wu et al. 2019). SPP-net sai parempia tunnistustuloksia sekä oli huomattavasti R-CNN:ää nopeampi testauksessa. Kuitenkin sen koulutuksessa kesti edelleen kauan ja se kärsi samoista optimointiongelmista kuin R-CNN.




*Kuva 15. SPP-net-arkkitehtuurista, kuva mukailtu Wu et al. (2019, kuva 4) pohjalta.*

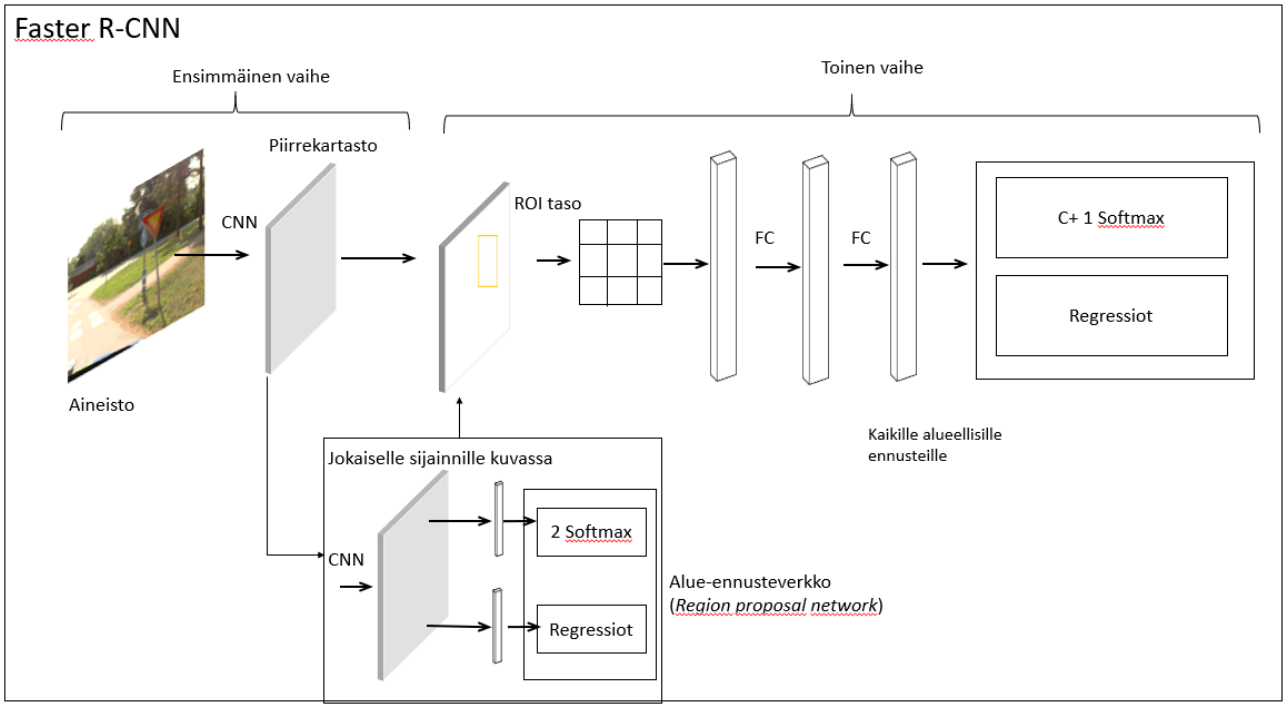
Fast R-CNN (kuva 16) kehitettiin vastaamaan SPP-net:in ja R-CNN:n kohtaamiin optimointiongelmiin. Siinä käytetään apuna SPP-net:in ominaisuuksia, mutta lisäksi tuodaan mukaan ROI-yhdistäminen (r*egion of interest*). Tämä auttaa SPP-net:in koulutusongelmissa (Du 2018, Wu et al. 2019). ROI irrottaa (*extract*) määritellyn pituisia osia tai ruutuja kohteista, jotka ovat ehdotetuissa (*proposed*) kuvissa eli kuva-alueilla, joissa kohde voi olla (Wu et al. 2019). Tämä nopeuttaa testausta ja koulutusta. Kuvassa 16 nähdään ROI-taso, joka ajetaan FC-tasojen kautta yhdistettyyn luokittelu- ja regressiovaiheisiin. Yhdistettynä nämä nopeuttavat algoritmin toimintaa entisestään (Du 2018). *Softmax*-taso ennustaa kohdeluokan ja *Regressor*-taso luokkakohtaiset *bounding box* -rajaukset kohteen paikallistamiseen (Liu et al. 2019). Fast-RCNN on testauksessa noin kymmenen kertaa ja koulutuksessa kolme kertaa nopeampi kuin SPP-net (Du 2018, Liu et al. 2019).




*Kuva 16. Fast R-CNN arkkitehtuurista kuva mukailtu Wu et al. (2019, kuva 4) pohjalta.*

Faster R-CNN (kuva 17) tuo mukaan oman erityispiirteensä eli RPN:n (*Region Proposal Network*), jolla ratkaistaan käsittelyä hidastava ongelma, *regional proposal problem* (Du 2018). RPN:n avulla

kohteen sijainti päätellään vasta lopullisesta piirreverkosta (*feature map*), kun kuva on mennyt konvoluutioiden (CONV) läpi. Tämä nopeuttaa tunnistusprosessia, koska resoluutio on pienempi ennustetuissa kohdeverkostoissa kuin alkuperäisessä kuvassa. RPN ehdottaa kohteita ikään kuin liukumalla kuvan yli, minkä pohjalta muodostetaan yhdeksän ankkuripistettä eri skaaloilla, leveyksillä ja korkeuksilla. Näistä pisteistä valitaan piirteet ja muodostetaan kohteen sijainnin määrittely regressioehdotuksen avulla.




*Kuva 17. Faster R-CNN arkkitehtuurista, kuva mukailtu Wu et al. (2019, kuva 4) pohjalta.*

RPN yhdistää sitten kohteet luokittelu- ja regressioehdotuksen perusteella, mitkä arvioivat nämä ankkuripisteet ja joko hylkäävät tai pitävät ne. Ankkureiden valintaan vaikuttaa niiden sijainti verrattuna muihin kohteisiin. Esimerkiksi ennustettujen kohteiden rajoilta ei valita ankkuria (Du 2018). Näiden pohjalta RPN valikoi noin 300 ankkuria yhden liukuman aikana. Kuten kuvassa 17 nähdään, nämä ehdotetut tunnistukset käsitellään vielä kertaalleen verkostossa, joka parantaa tunnistustulosta entisestään. Lopulta ennustettujen (*predicted*) piirteiden pohjalta muodostunut kuva käsitellään ROI-yhdistämisellä, joka nopeuttaa prosessointia muihin algoritmeihin verrattuna (Du 2018).

Faster R-CNN avulla on saatu lähes täydellisiä tunnistustuloksia, mutta sen toimintanopeus asettaa haasteita (Du 2018). Samoin pienten sekä eri skaaloissa olevien kohteiden tunnistaminen on haastavaa (Wu et al. 2019).

## 2.7 Konvoluutioneuroverkostot liikennemerkkien tunnistuksessa

### 2.7.1 Liikennemerkkien tunnistukseen käytettävät aineistot

Liikennemerkit ovat olleet varsin tyypillinen kohde kohteidentunnistuksessa ja niistä löytyviä avoimia aineistoja on runsaasti saatavilla. Näitä aineistoja ja algoritmien tunnistuskykyä niistä on testattu moneen otteeseen ja tulokset ovat olleet varsin hyviä. Aineistoja on koottu erilaisista kuvista, kuten pikselikooltaan pienistä 15 x 15 kuvista suuriin 2048 x 2048 pikselin kuviin (Temel et al. 2019). Tärkeää on myös aineistoissa annotoitujen liikennemerkkien koot, jotka vaihtelevat riippuen siitä, onko kuva otettu läheltä vai kaukaa. Aineistojen ominaisuuksissa on suurta vaihtelua niiden sisältämien kuvamäärien sekä esimerkiksi liikennemerkkiluokkien määrässä. Esimerkiksi CURE-TSD 2017 -aineistossa on keskimäärin 158 000 annotoitua kuvaa liikennemerkkiä kohden, kun taas tässä työssä koulutukseen käytettiin 700 annotoitua kuvaa. Joissain aineistossa näitä on taas huomattavasti vähemmän, kuten esimerkiksi GTSDB-aineistossa, jossa on keskimäärin 28 annotoitua kuvaa liikennemerkkiluokkaa kohden (Temel et al. 2019). Tämän perusteella voi arvioida tässä työssä koulutukseen ja testaukseen käytettävien kuvien olevan riittäviä toimivan algoritmin luomiseen.

### 2.7.2 Esimerkkejä tutkimusten tuloksista liikennemerkkien tunnistuksessa

Stallkamp et al. (2012) ovat käyneet läpi eri tutkimusten tuloksia liikennemerkkien havaitsemisessa vuosilta 2005–2010. Tunnistustarkkuus vaihtelee 85–97 % välillä, mutta näissä tutkimuksissa tunnistettavien luokkien määrät sekä luokittelun tarkkuus vaihtelevat suuresti. Vertailua hankaloittaa lisäksi käytettyjen aineistojen salaisuus, sillä niitä ei ole avoimesti saatavilla. Kyseinen artikkeli on laadittu vuonna 2012, jolloin nykyisenkaltaiset algoritmit eivät vielä olleet käytössä. Tämän myötä niitä ei voi suoraan verrata nykyisten algoritmien tuottamiin tunnistustuloksiin. Stallkamp et al. (2012) kehittivät myös oman aineistonsa GTRSB (*German Traffic Sign Recognition Benchmark*), joka on varsin tunnettu ja yleisesti käytetty algoritmien testaukseen. Aineistona oli yhteensä 50 000 kuvaa saksalaisista liikennemerkeistä 43 eri kategoriasta.

Aineiston testaukseen he käyttivät eri algoritmeja sekä vertasivat näiden toimintatarkkuutta ihmisten tunnistuskykyyn. Parhaaseen tulokseen päätyi CNN-yhdistelmä (*CNN committee*), joka oli jopa parhaiten suoriutunutta ihmistä parempi tunnistamaan kohteita. Oikein luokiteltujen kohteiden arvona (*Correct Classification Rate*) oli CNN-yhdistelmällä 99,46%, kun se parhaiten suoriutuneella ihmisellä oli 99,22 %. Ihmisen kyky tunnistaa merkkejä vaihteli riippuen liikennemerkkiluokasta. Esimerkiksi korkean nopeusrajoituksen liikennemerkit näyttivät olevan vaikeita ihmiselle, kun taas CNN-yhdistelmä tunnisti ne paremmin. Korkeissa nopeuksissa otetut kuvat olivat sumeampia, mikä vai-

keutti merkkien erottamista toisistaan. Erinäiset häiriöt merkeissä, kuten graffitit, aiheuttivat algoritmeille tunnistusongelmia, mutta eivät ihmisille. Algoritmien toiminta oli myös heikompaa sivuilta kuvattujen kohteiden tunnistuksessa, mikä taas sujui ihmisiltä paremmin. Esimerkiksi Zhu et al. (2016) kritisoivat kyseistä aineistoa liikennemerkkiluokkien vähäisyydessä (4 kpl) sekä mainitsevat liikennemerkkien koon olevan huomattavan suuri kuvissa.

Li et al. (2019) käsittelevät liikennemerkkien tunnistamista panoraamakuvilta ja kehittävät oman avoimen aineiston (Pano-RSOD) näiden pohjalta. Lisäksi he testaavat viiden eri kohteentunnistusalgoritmin toimintaa tällä aineistolla. Parhaiten kohteita yleisesti onnistui tunnistamaan YOLOv3, päästen AP (*average precision*) arvoon 71,83. Toisena oli Faster R-CNN (ResNet101) arvolla 70,56. YOLO oli myös selvästi nopein testatuista päästen 13 millisekuntiin per yksi kuva, kun toiseksi parhaiten suoriutunut Faster R-CNN käsitteli kuvat hitaammin (31.58 ms/kuva). Liikennemerkkien tunnistuksessa parhaan AP-arvon saavutti Faster-RCNN (63,59). YOLO sai tässä tapauksessa toiseksi parhaan tuloksen (61.53) ja huonoiten pärjäsi SSD (51,82).

Vertailtaessa tunnistusnopeutta ja tunnistustarkkuutta yhdessä havaitaan YOLOv3 saavan parhaat tulokset sekä olevan myös nopein (Li et al. 2019). Myös toinen yhden tason algoritmi SSD, on nopea tunnistamaan, mutta sen tarkkuus ei ole yhtä hyvä kuin muilla algoritmeilla. Kaikkein hitain on Faster RCNN (VGG-16), joka on arkkitehtuuriltaan monimutkaisempi kuin ResNet-101 pohjautuva Faster-RCNN.

Arcos-García et al. (2018) vertaavat omassa artikkelissaan eri algoritmeja liikennemerkkien tunnistuksessa. Heidän vertailussaan YOLO:ssa käytettiin toista versiota, joten se ei ole suoraan verrattavissa kolmannen version toimintaan. Heidän tuloksissaan Faster R-CNN (*Inception Resnet V2*) toimi huomattavasti paremmin kuin YOLOv2 usealla eri osa-alueella. YOLO:n nopeus oli kuitenkin moninkertaisesti parempi (46,55 fps) kuin Faster R-CNN (2,26 fps). Samalla muistin käyttö oli noin 14 kertaa vähäisempää YOLO:n avulla verrattuna Faster R-CNN:ään. Varsinkin reaaliaikaisen tunnistuksen mahdollistamiseksi kuvataajuuden olisi oltava mahdollisimman korkea (Arcos-García et al. 2018). AP-arvoilla mitattuna Faster R-CNN pääsi liikennemerkkiluokasta riippuen yli 93,5:een, kun taas YOLO:n vaihteluväli oli 65–88 välillä. Samalla keskiarvoinen IoU-arvo liikkui Faster R-CNN avulla 89–91 välillä, kun se YOLO:lla vaihteli välillä 73–75.

On hyvä huomata, että tutkimustulosten vertailu on hankalaa erilaisten mittarien ja aineistojen vuoksi, joten suoraa päätelmää algoritmien paremmuudesta tutkimusten välillä on vaikeaa tehdä. Esimerkiksi Li et al. (2019) ja Arcos-García et al. (2018) tutkimuksissa tunnistuskykyä on mitattu AP ja mAP avulla, joita ei käytetty Stallkamp et al. (2012) artikkelissa. Toisaalta Li et al. (2019) eivät käsitelleet

tarkkuutta samalla tavalla. Tunnistuksen tavoitteet voivat myös erota toisistaan. Esimerkiksi tunnistus ja luokittelu ovat tehtävinä erilaisia ja näiden onnistumista voidaan mitata myös erikseen. Myös liikennemerkkien jaottelu eri liikennemerkkiluokkiin voi olla tavoitteena. Esimerkiksi nopeusrajoituksesta kertovat merkit muistuttavat toisiaan ja näiden erottelu toisistaan voi olla tutkimuksen tavoitteena (Stallkamp et al. 2012).

Monissa eri tutkimuksissa todetaan liikennemerkkien tunnistuksen ja luokittelun olevan jo todella tarkkaa. Esimerkiksi Zhu et al. (2016) tutkimuksessa kuvataan, että GTSRB-aineistosta on onnistuttu tunnistamaan 100 % liikennemerkeistä ja luokittelemaan näistä 99,67 %. Hienosen (2014, 84) työssä lopputuloksena taas oli 96 % tunnistustarkkuus liikennemerkeistä ja 98,55 % tarkkuus näiden luokittelussa.

### 2.7.3 Algoritmien eroavaisuudet

Kuten aiemmin mainittua, algoritmien toimintakyvyn vertailuun käytetään yleensä yhtä yhteistä standardisoitua aineistoa, kuten Pascal VOC tai ImageNet ILSVRC. Näiden aineistojen tyypillisiä piirteitä on esimerkiksi se, että tunnistettavat kohteet ovat muun kuvan kokoon verrattuna suuria (Zhu et al. 2016). Esimerkiksi panoraamakuvilta tehtävään kohteentunnistukseen tämä vaikuttaa negatiivisesti, sillä liikennemerkit ovat pieniä verrattuna kuvaan muuten (Zhu et al. 2016). Myös Li et al. (2019) tunnistavat nykyisten algoritmien sopivan huonosti panoraamakuviin, joiden kohteet voivat olla vääristyneitä ja joissa niitä on usein enemmän kuin perinteisissä kuvissa. Heidän Pano-RSOD aineistonsa pyrkimyksenä onkin nopeuttaa näiden algoritmien kehitystä

YOLO:n ero muihin algoritmeihin voidaan tiivistää toimintatavan eroavaisuuksiin. Muut algoritmit perustuvat ehdotettujen kohteiden tunnistamiseen kuva-alueittain ja näiden luokitteluun (*region proposal-based object detection*), kun YOLO taas ennustaa rajatut alueet (*bounding box*) sekä tiettyyn luokkaan kuulumisen todennäköisyyden kerran kuvan nähtyään (Du 2016, Liu et al. 2019, Putra et al. 2018, Wu et al. 2019). Koska tämä kaikki tapahtuu YOLO:ssa putkessa (*pipeline*) yhdessä verkossa, se on huomattavasti nopeampi kuin muut algoritmit ja pystyy saavuttamaan jopa 155 kuvataajuuden fps (*frames per second*) (Wu et al. 2019).

Eri tutkimusten tulokset algoritmien vertailusta vaihtelevat runsaasti. Esimerkiksi Arcos-García et al. (2018) tutkimuksessa Faster R-CNN pärjäsi tunnistuksessa huomattavasti YOLO:a paremmin, mutta nopeudessa taas ei. Myös Du (2018) vertailee kohteen tunnistusta R-CNN ja YOLO-verkoston avulla. Hän kuvailee, että eri R-CNN algoritmien avulla tehtävä tunnistaminen on yleisesti hitaampaa kuin YOLO:n avulla (Du 2018, Li et al. 2019). Vaihtelevasti tunnistustarkkuus on kuitenkin parempi esimerkiksi Faster R-CNN avulla (Li et al. 2019), mutta ei aina.

Useassa artikkelissa (Arcos-Garcia et al. 2018, Zhu et al. 2016) havaitaan kuvassa olevan liikennemerkin koon olevan merkittävä tekijänä tunnistustarkkuudessa. Mitä isompana merkki kuvassa näkyy, sitä tarkemmin se onnistutaan tunnistamaan (Arcos-Garcia et al. 2018). Zhu et al. (2016) myös huomasivat yhden vaiheen tunnistusalgoritmien, kuten YOLO:n ja SSD:n suoriutuvan keskimäärin heikommin pienten merkkien tunnistamisessa verrattuna kahden vaiheen tunnistusalgoritmeihin.

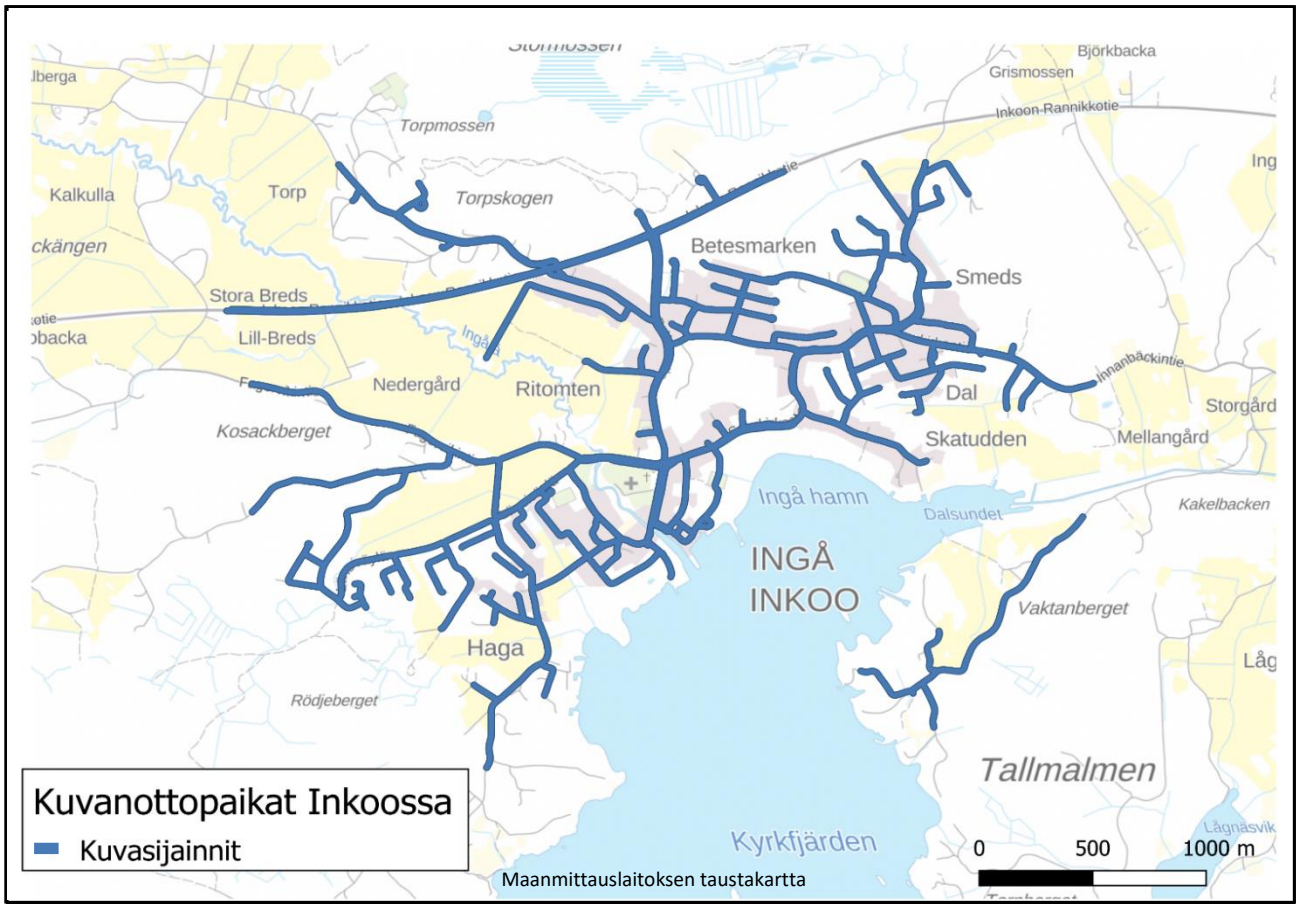
Reaaliaikainen tunnistaminen on mahdollista lähinnä yhden tason algoritmeilla. Vaikkakin YOLO voi saavuttaa 155 fps tunnistusnopeuden, niin yleisempää toimintakykyä kuvastaa kuitenkin noin 50-60 kuvataajuus YOLO:n ja noin 10-20 kuvataajuus esimerkiksi Faster R-CNN algoritmilla (Li et al. 2019). Täten voi todeta YOLO:n olevan huomattavasti nopeampi kuin kahden vaiheen tunnistusalgoritmit. Algoritmin valinnassa olisi tunnistustarkkuuden ja nopeuden lisäksi huomioitava kuitenkin myös laskentatehon tarve, etenkin GPU:n muistin osalta sekä mallien implementointiin kuluva aika, joka vaihtelee merkittävästi algoritmin mukaan (Arcos-García et al. 2018)

YOLO:n sanotaan tuottavan vähemmän vääriä positiivisia havaintoja taustasta kuin sen kilpailijat. YOLO käyttääkin vähemmän *bounding box* -rajauksia eli ennustuksia kuvaa kohden kuin *Selective search* -metodia käyttävät R-CNN-algoritmit (Liu et al. 2019). Sen huonoimpana puolena on usean pienen kohteen tunnistaminen ryhmästä (Du 2018, Redmon et al. 2016). Tätä on kuitenkin parannettu ensimmäisestä versiosta, joka sisälsi oletuksen, että yhdessä solussa voisi olla enintään kaksi kohdetta. Tämä heikensi varsinkin runsaskohteisten kuvien käyttämistä testauksessa (Wu et al. 2019). Tämän vuoksi YOLO:n versioissa kaksi ja kolme otettiin mallia toisesta yhden tason avulla toimivasta algoritmista, *Single-Shot Mulibox Detector*:sta (SSD) (Wu et al. 2019).

# 3. Aineisto

## 3.1 Panoraamakuvat

Tämä tutkimus pohjautuu panoramakuvien käyttöön. Tyypillisesti panoraamakuvaksi kutsutaan 100–360° laajuisten kuvakulmien kuvausta (Kauhanen & Rönnholm 2012). Tässä työssä aineistona käytettävät panoraamakuvat on otettu Inkoon alueelta (kuva 18) kesällä 2018 osana Maanmittauslaitoksen Uudet kuvanmittausteknologiat -projektia, jolla pyrittiin selvittämään erilaisia käyttökohteita kuville.




*Kuva 18. Panoraamakuvien ottopaikat Inkoossa, tausta Maanmittauslaitoksen taustakarttasarja.*

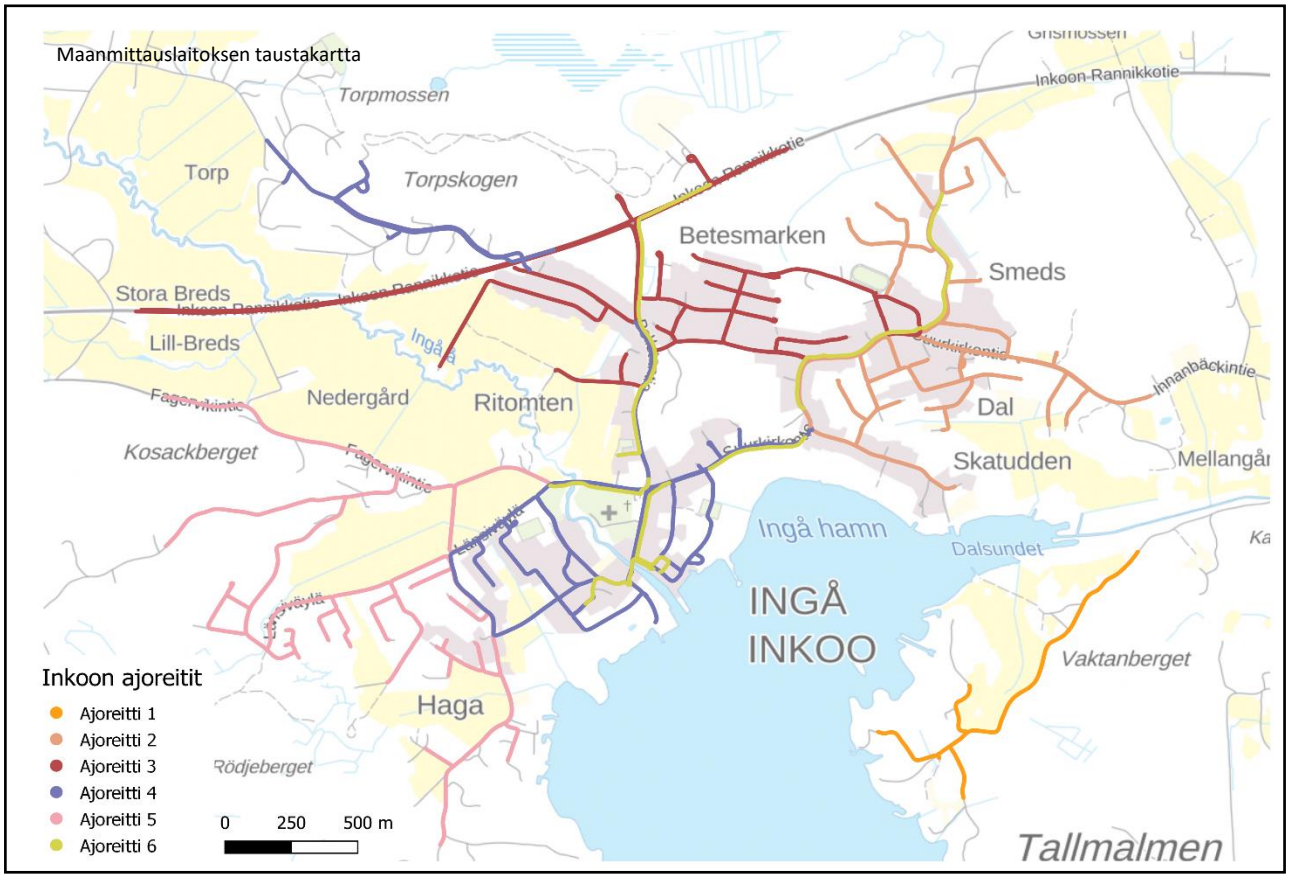
Kuvat on otettu Trimble MX7 -mobiilikartoitusjärjestelmällä, joka tallentaa kuvien lisäksi kuvanottohetken, sijainnin, kulman, kulkusuunnan, korkeuden, kallistuman ja kierron. Panoraamakuvien sijainti- ja orientointitiedot saadaan mobiilikartoitusjärjestelmän GNSS-vastaanottimen ja IMU-yksikön (*Inertial Measurement Unit*) avulla. Panoraamakuvat on otettu 3 metrin välein pohjautuen GNSS-sijaintiin. Tämän avulla saadaan jatkuvaa tietoa kuvausauton sijainnista ja täten kuvanottopaikkojen koordinaateista. Mobiilikartoitusjärjestelmä sijaitsee kuvausauton päällä ja otetut kuvat

ovat tarkkuudeltaan 30 megapikseliä. Kuvassa 19 näytetään esimerkki Inkoon panoraamakuvista sekä kuvataan kuvan sisältämät metatiedot, kuten koordinaattisijainti ja kuvauskulma.

| gps_seconds[s] | panorama_file_name | latitude[deg] | lognitude[deg] |
|---|---|---|---|
| 1219569960.95225 | pano_0000_000000 | 60.044406767276 | 24.0378533347547 |
| roll[deg] | pitch[deg] | heading[deg] | altitude_ellipsoidal[m] |
| 0.758887077004063 | 4.7765985379083 | 31.9235901257789 | 23.13422586862 |

*Kuva 19. Esimerkkikuva panoraamasta ja sen sisältämistä metatiedoista.*

Kamerasensoreita on yhteensä kuusi, joista yksi on suunnattu ylöspäin. Kamerasensoreiden kuvista yhdistetään panoraamanäkymä, jonka peittävyys kuvasta on 90 % luokkaa täydestä pallopanoraamasta (Geotrim esittelymateriaali 2018). Ajoradan sijainti on laskettu jälkilaskennassa Applanix POSPac MMS -ohjelmistolla. Inkoon eri reiteiltä kertyi panoraamakuvia yhteensä 16 432 kappaletta ja reittejä oli kuusi erilaista (kuva 20). Osa kuvista on otettu samoilla reiteillä kääntymisten vuoksi, jonka takia osa kartan pisteistä on päällekkäisiä




*Kuva 20. Inkoon eri ajoreitit numeroituna.*

Panoraamakuville on mainittu useita eri käyttökohteita, kuten rekisterien ja tietokantojen ylläpito sekä vapaan tilan mittaaminen, minkä pohjalta on esimerkiksi mahdollista arvioida, kuinka iso ajoneuvo mahtuu kulkemaan sillan ali. Muita kohteita ovat esimerkiksi ulkomainonnan inventointi sekä kohteiden kartoittaminen (Geotrim esittelymateriaali 2018). Uudet kuvanmittausteknologiat -projektissa toteutettiin myös kuntahaastatteluja, joissa ilmeni monien kuntien olevan kiinnostuneita panoraamakuvien hankinnasta. Kunnissa on tunnistettu käyttökohteiksi esimerkiksi kiinteistönmuodostus, rakennusvalvonta, kadunrakennus ja liikenteen suunnittelu, kaavoitus sekä ympäristöpalvelut. Myös kohteiden seuranta ja valvonta sekä infraomaisuuden kartoitus ja hallinta ovat tunnistettuja käyttötapauksia kunnissa (Kuntahaastattelujen yhteenveto 2019).

Myös Salo (2018) kertoo panoraamakuvien yleiseksi käyttökohteeksi erilaiset tieomaisuuserien inventoinnit. Yleensä nämä koostuvat liikennemerkeistä, pysäkeistä, mainostauluista sekä maalimerkinnöistä (Salo 2018, 31). Kuvien todetaan auttavan myös tilanteissa, joissa on tarpeellista suorittaa jälkimittauksia tai varmistaa tiedon pätevyys maastokäynnin jälkeen. Tämä jo itsessään säästää kustannuksia, kun ylimääräisiltä maastokäynneiltä vältytään.

Panoraamakuvien käyttöä kohteentunnistuksessa ovat pyrkineet edistämään esimerkiksi Li et al. (2019) ja Zhu et al. (2016) julkaisemalla koulutukseen ja testaukseen uusia panoraamakuviin perustuvia aineistoja. Panoraamakuvat ovat mielenkiintoinen tutkimuskohde niiden kohderunsauden vuoksi. Suurin ero Li et al. (2019) julkaiseman PanoRSD-aineiston ja muiden tarjottavien aineistojen välillä onkin juuri yhdeltä kuvalta saatavien kohteiden määrä, joka panoraamoissa on keskimäärin suurempi kuin muissa kuvatyypeissä (Li et al. 2019). Pikselimäärät määrittävät kuvan tarkkuuden ja suuremman pikselikoon kuvilta on ihmisen helpompaa tunnistaa kohteita. Käytettävät algoritmit eivät kuitenkaan mukaudu samaan malliin, vaan pikselikoon suuruus vaikeuttaa ja hidastaa merkittävästi algoritmeja ja niiden toimintaa valitusta menetelmästä riippuen. Tästä syystä tässä työssä verrattiin tunnistuksen tarkkuuttaa panoraamakuvien ja niistä luotujen osakokonaisuuksien välillä, jotta nähdään, kuinka suuri merkitys kuvakoolla on.

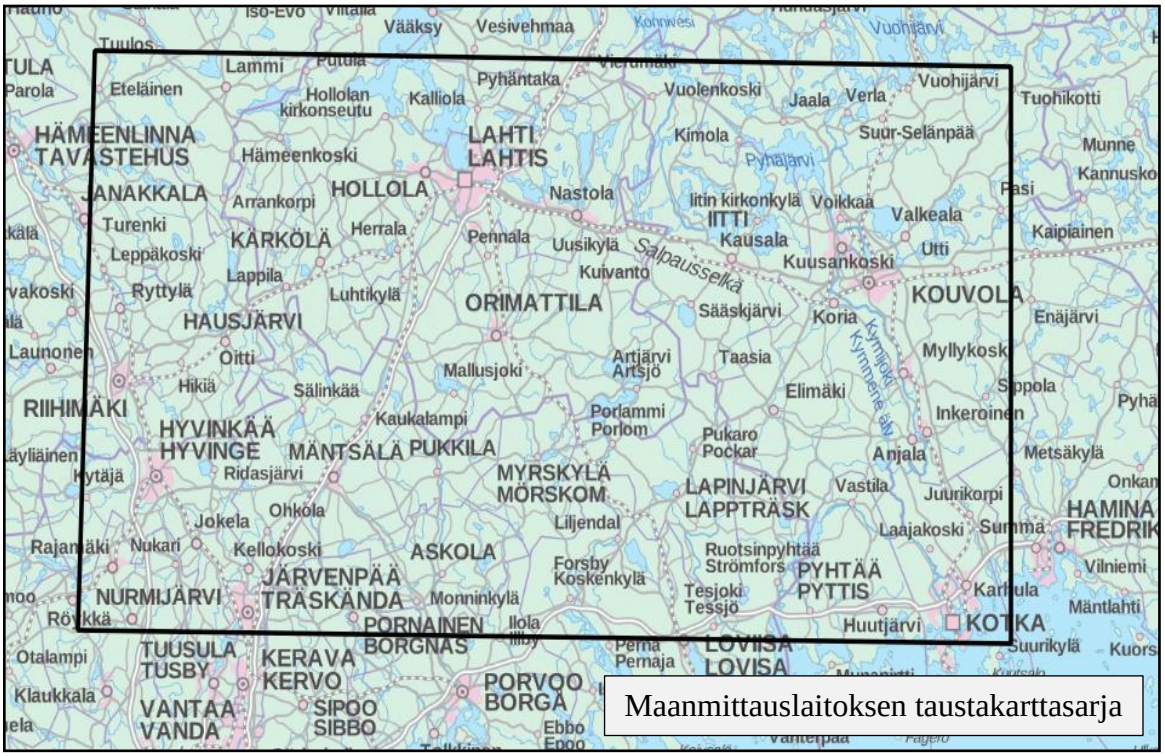
Panoraamakuvien käytöstä on syntynyt myös paljon keskustelua, sillä ne voivat sisältää henkilötietona pidettäviä asioita sekä loukata ihmisten yksityisyyttä (Nodari et al. 2012). Esimerkiksi teknologiayhtiö Googlen Street View -palvelusta on voinut löytyä arkaa materiaalia, jota Nodari et al. (2012) pyrkivät anonymisoimaan peitteillä kuvista. Koska tämän opinnäytetyön kuvat sisältävät mahdollisesti myös arkana pidettyä aineistoa, työssä esitellään vain niitä kuvia, joissa mahdollisia loukkauksia ei esiinny. Vaikka kuviin on tehty automaattista peittoa esimerkiksi autojen ja ihmisten osalta, voi niistä toisinaan erottaa rekisterinumeroita ja ihmisten kasvoja.

## 3.2 Mapillaryn kuvat

Mapillary on IT-alan toimija, jolle vapaaehtoiset toimittavat kuva-aineistoja yleensä katutason näkymistä. Nämä kuvausaineistot kerätään joukkoistamalla ja aineistoja muokataan Mapillaryn omissa prosesseissa esimerkiksi peittämällä henkilötiedot. Mapillarylta on mahdollista saada aineistoa tutkimuskäyttöön ja niitä voi pyytää heidän rajapintansa sisältämistä luokista (Mapillary 2020). Tämän lisäksi tarkennetaan pyydetyistä kuvista pikselikoko, määrä, formaatti sekä toimitusosoite. Tässä työssä tutkittiin kärkikolmioita, jotka löytyivät kyseisestä rajapinnasta.

Kuva-aineisto kattoi yhteensä 500 kuvaa, jotka toimitettiin kahdessa osassa. Ensimmäinen kuva-aineisto osoittautui virheelliseksi, sillä sen aluerajaus oli liian pieni. Sen sisältä ei löytynyt tarpeeksi kuvia, sillä suuri osa kuvista oli Venäjältä. Näiden ongelma oli kärkikolmioiden väärä muoto sekä väritys. Osa kuvista oli puolestaan Ruotsista, mutta koska Ruotsin kärkikolmiot vastaavat melko hyvin Suomen kärkikolmioita, eivät nämä tuottaneet ongelmia. Toisella kerralla pyydettiin kuvia laa-

jemmalta alueelta Suomesta (kuva 21). Niiden kuvakoko oli hieman pienempi kuin aiemmassa toimituksessa. Pyydetty alue sijaitsi muualla kuin Inkoossa, jotta testi- tai koulutusaineistoon ei sekoittuisi samoja kärkikolmioita eli aineisto olisi riippumaton.




*Kuva 21. Mapillarylta pyydetyn toisen toimituskerran bounding box -rajaus.*

# 4. Menetelmät

## 4.1 Käytetyt ohjelmistot ja menetelmät

### 4.1.1 Azure-ympäristö ja YOLO:n tarvitsemat komponentit

Lähtökohtana työssä oli toteuttaa YOLO:n versio kolmea hyödyntävä kohteentunnistusalgoritmi, joka voisi tunnistaa valittuja infrastruktuurikohteita automaattisesti kuvilta. YOLO:n versio kolme valittiin tunnistustyökaluksi, koska sen todettiin olevan yksinkertaisimmin implementoitavissa tähän työhön. Lisäksi sen tunnistustarkkuus ja nopeus ovat osoittautuneet erittäin kilpailukykyiseksi verrattuna muihin algoritmeihin. Käytännön asioihin löytyi myös monipuolisesti dokumentaatiota sekä asennusohjeita, joita hyödyntää käyttöönotossa.

Algoritmin käytössä hyödynnettiin Microsoft Azuren ja Esrin yhteistyönä kehittämää virtuaalitietokonetta varustetulla GeoAI-DSVM (*Geo AI Data Science Virtual Machine*) Windows 10 -käyttöjärjestelmällä. Pilvipalvelu toimi alustana kuvien hallinnoinnille, tarkastelulle sekä analyysille. Virtuaalikoneen suurin hyöty tuli kuitenkin sen GPU:n (Tesla K80) laskentatehosta, jolla algoritmi koulutettiin. Koulutus vaati koneelta paljon laskentatehoa ja pelkän CPU:n hyödyntäminen ei ajallisesti ollut järkevää. On vaikea arvioida, paljonko laskentatehoa koneoppimisalgoritmin koulutus vaatii, sillä se riippuu käsillä olevasta tehtävästä. Esimerkiksi Stenroos (2017) toteaa monien tutkijoiden käyttävän Tesla K40-GPU:ta koulutuksessa. Pilvipalveluiden käyttö big datan hallinnointiin, käsittelyyn sekä analyyseihin on kasvattanut suosiotaan jo vuosien ajan (Guo 2020, 348). Uusia palveluntarjoajia on tullut markkinoille, mutta yleensä Microsoft Azure ja AWS (*AmazonWebServices*) mainitaan tarjoajina.

Kärkikolmiot valittiin tunnistettaviksi kohteiksi, sillä niitä oli oman aineiston joukossa runsaasti. Tämän myötä aineistoa testaukseen ja koulutukseen olisi riittävästi. Kärkikolmiot ovat ainutlaatuisen muotonsa puolesta myös suhteellisen erottuvaisia verrattuna moneen muuhun liikennemerkkiin, mikä helpottaa tunnistusta. Muissakin tutkimuksissa liikennemerkit ovat olleet suosittuja tunnistuskohteita (esim. Arcos-Garcia et al. 2018, Li et al. 2019, Hienonen 2014, Stallkamp et al. 2012, Temel et al. 2019). Työ aloitettiin siirtämällä kuva-aineisto (16 432 kuvaa) virtuaalikoneelle sekä asentamalla YOLOv3:n toimintaa tukevat ohjelmat. Näitä ohjelmia ja kirjastoja olivat:

1. LabelImg
    - Käytetään koulutusaineiston annotointiin (eli *bounding box* -rajausten luomiseen).
2. Darknet

- Darknet on avoimen lähdekoodin neuraalisten verkkojen viitekehys, jota käytetään YOLO-verkoston implementointiin. Tässä työssä hyödynnettiin Windows 10 -käyttöjärjestelmää.

3. CMake (3.16.2-win64-x64)
    - Käytettiin darknet.exe-tiedoston luomiseen, parametrien asettamiseen sekä arkkitehtuurin rakentamiseen Microsoft Visual Studiossa.
4. OpenCV (3.4.9 MS Visual studio 2017)
    - Avoimen lähdekoodin konenäkö- ja koneoppimiskirjasto, joka tukee Darknetin prosesseja.
5. GIT
    - Kirjastojen lataus ja versionhallinta.
6. Anaconda (3.0)
    - Python-ohjelmointikielen kirjastojen hallintaan sekä komentojen suorittamiseen.
7. CUDA (v10.2)
    - Nvidian ohjelmointirajapinta, jota tarvitaan CPU/GPU laskennassa.
8. cuDDN (7.6)
    - CUDA:n alainen kirjasto, jolla toteutetaan neuroverkkojen syväoppiminen GPU-avusteisesti.
9. Microsoft Visual Studio 2017 (Visual Studio Toolkit)
    - Käytetään arkkitehtuurin luomisessa ja parametrien muunnoksissa.
10. YOLOv3-kirjasto Githubista
    - YOLO-verkoston koodit.

Mainitut ohjelmistot toimivat yhteydessä toisiinsa ja ne ovat välttämättömiä algoritmin luomiseen. Yhteys ohjelmien välillä varmistetaan asettamalla ympäristömuuttujat (*environmental variables*) yhteneväisiksi. Ohjelmistojen asennuksessa, Azure-ympäristön käyttöönotossa sekä YOLO:n implementoinnissa saatiin konsulttiapua TietoEVRY:n datatieteilijä Iftikhar Ahmadilta.

### 4.1.2 Aineiston manuaalinen tarkastelu ja jaottelu osakokonaisuuksiin

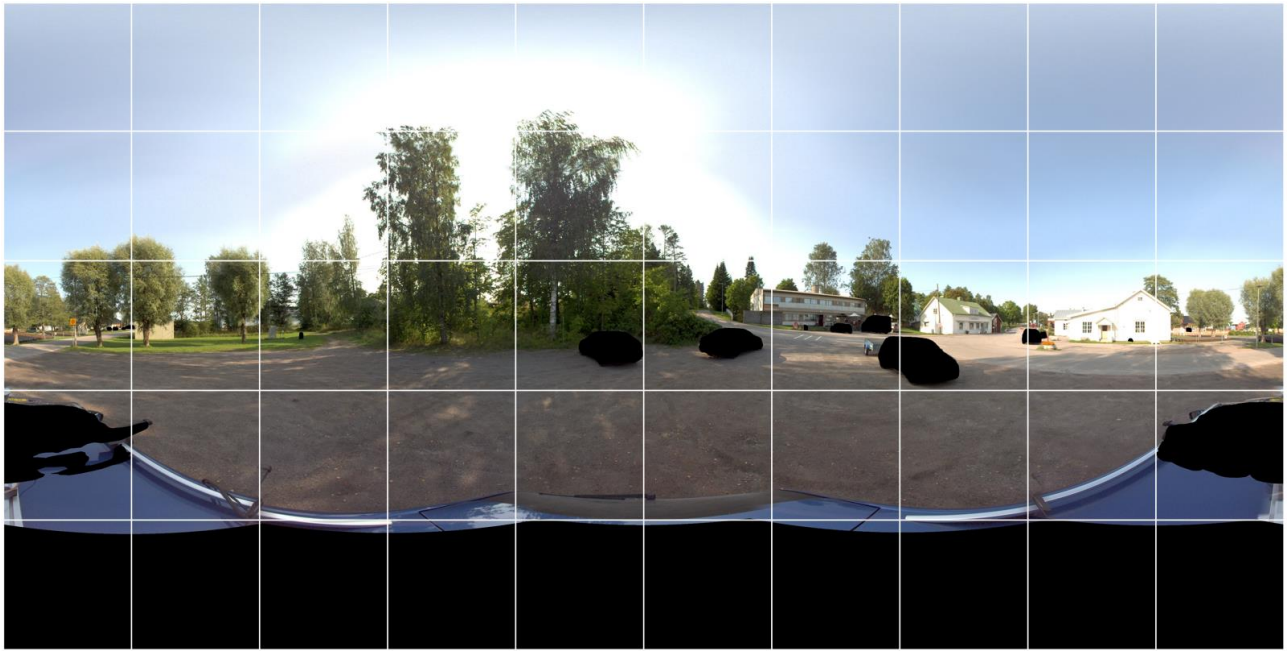
Liikennemerkkien tunnistusta lähdetään usein rakentamaan määrittämällä eri kohteet luokkiin niiden toimintaluokan mukaan. Tämän jälkeen ne voidaan vielä luokitella muodon mukaan. Esimerkiksi samanmuotoiset merkit, kuten pystykolmiot, määriteltäisiin omaksi luokakseen (Zhu et al. 2016). Tässä tapauksessa koulutusta varten valittiin vain kärkikolmiot, joten ne jaoteltiin vain yhteen luokkaan.

Kohteentunnistusalgoritmin koulutusta varten on luotava sopivaa aineistoa, jonka pohjalta algoritmi tietää miltä haluttu kohde näyttää. Tämä onnistuu annotaatioiden eli *bounding box* -rajausten kautta.

Koko Inkoon kuva-aineisto (16 432 kuvaa) käytiin ensin manuaalisesti läpi havainnoiden ajoreitiltä ryhmiä, joista oli mahdollista havaita kärkikolmio. Yhteensä ajopisteitä, joista näitä havaintoja oli mahdollista nähdä, kertyi 1287. Yhdestä kärkikolmiosta sai yhden ohituksen aikana noin 10-30 kuvaa riippuen esimerkiksi siitä, oliko kolmio käännöksen yhteydessä vai pidemmän suoran varrella. Annotointien merkinnässä huomioitiin Li et al. (2019) kommentit, joiden mukaan annotoinnissa koko merkki pitää saada suorakulmarajauksen piiriin.

Annotointi tehtiin yhteensä kaksi kertaa, koska ensimmäisellä kerralla se toteutettiin panoraamakuville, jotka havaittiin liian kookkaiksi koulutukseen. Toisella kerralla panoraamakuvista pilkottiin osakokonaisuuksia, jotta ne sopisivat YOLO:n asettamiin parametreihin paremmin. Tämä toteutettiin Patches.py -koodin (liite 1) avulla, mikä loi 50 osaa yhdestä panoraamakuvasta ja kuusi osaa yhdestä Mapillaryn toimittamasta kuvasta. Patches.py vaatii os-, glob- ja PIL-python-kirjastot toimiakseen.

YOLO:a asennettaessa valitaan verkostomallin koko, joka vaihtelee kuvakoon mukaan. Esimerkiksi suurin koko verkostolle on 608 x 608, mutta tämän prosessointi on raskasta. Tämän myötä päädyttiin yleiseen YOLOv3-416 -verkostomalliin, jota voidaan hyödyntää 832 x 832 kuvilla. Inkoon panoraamakuvien koko oli alun perin 8000 x 4000 pikseliä, joten niistä luotiin pilkottuja kuvakokonaisuuksia 832 x 832 pikselikokoon, jotta toimintavarmuus YOLO:a hyödyntäen olisi parempi. Esimerkki kuvan jakautumisesta näihin osakokonaisuuksiin löytyy kuvasta 22.



*Kuva 22. Esimerkki kuvan (8000 x 4000) jaottelusta pilkottuihin osakokonaisuuksiin (832 x 832).*

Valitun verkostomallin mukaan YOLO pienentää annetut kuvat 416 x 416 kokoon, jotta ennustusten teko olisi luotettavaa. Verkostomallin koon saa ylittää kaksinkertaisesti, jotta tunnistustaso säilyy luotettavana. Verkostomalli myös määrittää konfiguraatiotiedoston (liite 4), johon tutustutaan myöhemmin.

Koska kuvien jaottelussa ei huomioitu kärkikolmioiden sijaintia, osa niistä leikkautui (kuva 23). Alussa määriteltiin, että kolmiosta leikkaantuessa pois yli 30 %, on panoraaman jaottelu tehtävä manuaalisesti, jotta koulutusaineisto olisi parempaa. Itsessään leikkaus ei aina haittaa, koska koulutusaineiston on tarkoituskin olla mahdollisimman monipuolista ja vastata reaalimaailman ongelmia, kuten kuvakohteiden vääristymiä. Yhteensä näitä manuaalisesti leikattavia kohteita kertyi 20. Useassa kuvassa oli enemmän kuin yksi kärkikolmio, joten kuvia yksittäisistä kärkikolmioista kertyi 26 enemmän kuin alkuperäisiä kuvia oli. Panoraamoja leikattiin yhteensä 257 kappaletta ja Mapillaryn toimittamia kuvia 417 kappaletta, eli yhteensä 674 kuvaa. Viimeiseen lukuun lisätään vielä mainitut 26 kuvaa, joissa kärkikolmioita oli enemmän kuin yksi, minkä myötä koulutusaineistoksi kertyi yhteensä 700 kuvaa.

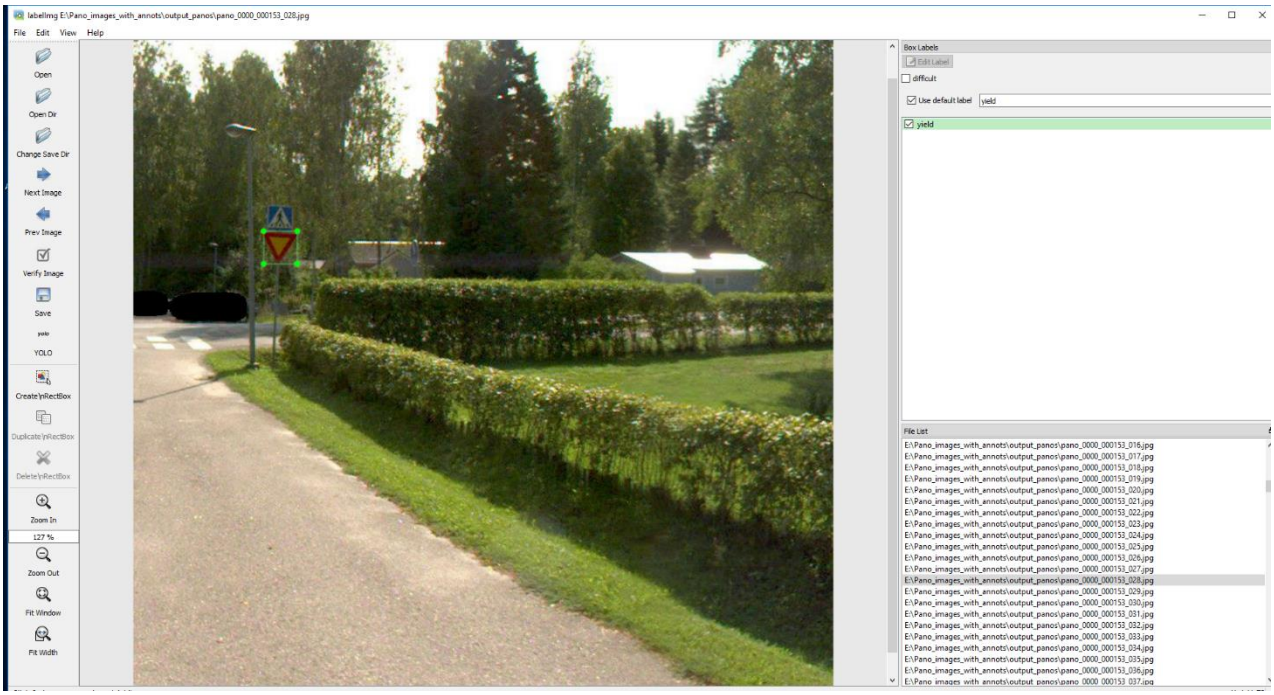

*Kuva 23. Esimerkkikuva Mapillaryn aineistosta, jossa liikennemerkki on vasemmassa kuvassa leikkaantunut liikaa. Oikealla puolella uudelleen manuaalisesti jaoteltu kuva.*

### 4.1.3 LabelIMG -suorakaiderajauksien teko

Kuvien laatutarkistuksen jälkeen siirryttiin annotointeihin. Nämä toteutettiin LabelIMG-ohjelman avulla (kuva 24). Annotaatioiden tekoa ennen piti määrittää kohdeluokat, joihin rajaukset kohdistuvat. Luokkana käytettiin ”*yield*” eli kärkikolmioluokkaa, mutta luokkia voi olla useampiakin. Ohjelmassa on mahdollista valita kahden eri annotaatiotyypin välillä: 1. YOLO tai 2. PascalVOC (kuva 25). Tässä työssä hyödynnettiin YOLO-muotoisia annotaatioita, jotka kuvataan tekstimuodossa:

”0 0.388221 0.425481 0.026442 0.024038”

jossa ensimmäinen numero ”0” viittaa luokkaan ja muut numerot (koordinaatit) kuvaavat etäisyyttä kuvan keskipisteestä (0.0), jonka perusteella *bounding box* -rajaus on määritetty. Yhteensä annotaatioita kertyi 700. PascalVOC taas on monimutkaisempi XML-tiedosto, joka sisältää samat tiedot hieman eri muodossa, minkä lisäksi siinä määritellään tarkempi tiedostopolku (kuva 25).




*Kuva 24. Esimerkkikuva LabelIMG:n käyttöliittymästä, jossa annotaatiot tehdään*

```
<annotation>
    <folder>Copy_5_imgswith_yields</folder>
    <filename>pano_0000_000025_033.jpg</filename>
    <path>E:\Inkoo_5_only_yields\Copy_5_imgswith_yields\pano_0000_000025_033.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>832</width>
        <height>832</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>yield</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>172</xmin>
            <ymin>160</ymin>
            <xmax>269</xmax>
            <ymax>337</ymax>
        </bndbox>
    </object>
</annotation>
```

*Kuva 25. Pascal VOC-annotaation tietosisältö, jossa spesifioidaan tiedoston sijainti, kuva-koko, annotaatioluokka, sekä bounding box -rajaus kuvilta.*

## 4.2 Algoritmin koulutus

LabelIMG-työkalun pohjalta luoduista 700 annotoidusta kuvasta valittiin 540 kuvaa koulutusaineistoksi ja 160 kuvaa koulutuksen validointiin. Kuvat pyrittiin valikoimaan siten, että niissä olisi mahdollisimman paljon variaatiota taustojen mukaan. Tämä auttaa koulutuksen monipuolistamisessa sekä vähentää ylisovittamisen riskiä. Mapillaryn kuvien ja Inkoosta otettujen panoraamakuvien välillä on myös suurta vaihtelua. Kuten aiemmin on todettu, osa Mapillaryn kuvista on otettu Ruotsista sekä erilaisilta seuduilta kuin vastaavat panoraamat.

Darknet-kirjastoa hyödyntäen YOLO:n koulutus vaatii neljä tiedostopolkua sekä luokiteltavien kohteiden määrän, jotka kootaan Data-tiedostoon:

1. classes = luokkien määrä
2. train = koulutukseen käytettävien kuvien sijainnit, joista löytyy myös annotaatiot
3. valid = koulutuksen validointiin käytettävien kuvien sijainnit, joista löytyy myös annotaatiot
4. classes = annotaatioiden nimi, kuten tässä tapauksessa ”*yield*” eli kärkikolmio
5. backup = kansio, johon tallennetaan koulutuksessa syntyvät painoarvotiedostot.

```
classes= 1

train  = C:\darknet-master\darknet-master\build\darknet\x64\data\obj\newtxt\trainimg.txt

valid  = C:\darknet-master\darknet-master\build\darknet\x64\data\obj\newtxt\validimg.txt

names  = C:\darknet-master\darknet-master\build\darknet\x64\data\nls.names

backup = C:\darknet-master\darknet-master\build\darknet\x64\data\backup\
```

Train ja valid ovat tekstimuotoisia tiedostoja, jotka sisältävät pelkästään tiedostopolun käytettävään kuvaan. Esimerkiksi tiedostopolun saa selville komentokehotteella ”dir /s/b *.jpg >listjpg.txt” kun on oikeassa tiedostokansiossa. Koulutusta testattiin ensin CPU:n (*Central Processing Unit*) avulla, mutta se osoittautui hyvin hitaaksi. CPU-avusteisena koulutukseen kului 20 tuntia ja tämä onnistui ajamaan tässä ajassa vain 40 iteraatiokierrosta. Tämän takia laskennassa hyödynnettiin GPU:ta, joka pystyi tuottamaan vastaavat 40 iteraatiokierrosta 15 minuutin aikana.

Laskenta vaatii parametrimuutoksia konfiguraatiotiedostoon (YOLOv3.cfg, liite 4), joka ladataan Darknet-kirjaston mukana (Github -Darknet 2020). CFG-tiedoston määrittelyssä käytettiin apuna samaa AlexeyAB:n Github-sivua (AlexeyAB 2020), jossa parametrejä muutettiin valittujen luokkien, kuvakoon, tarkkuusluokituksen, iteraatiokierroksien ja filttereiden mukaan. Näin CFG-tiedosto kustomoidaan sopimaan oman datan käsittelyyn. Käytettäessä GPU:ta laskentaan, myös Makefile:n parametreja pitää muuttaa. Tässä tiedostossa asetetaan Darknet käyttämään GPU:ta CPU:n sijaan, sekä sallitaan CUDA:n ja CUDDN:n hyödyntäminen laskennassa. Lisäksi ladataan ”darknet53.conv.74” painoarvotiedosto, jota käytetään koulutuksessa apuna. Se on valmiiksi jo koulutettu tunnistamaan piirteitä ja tämän myötä myös kohteita. Uudessa koulutuksessa käytetään sen arkkitehtuuria hyödyksi ja asetetaan oletuskohteiden sijaan se tunnistamaan kärkikolmioiden piirteitä.

Tämän jälkeen kaikki on valmista koneoppimisen aloittamiseen. Komentona syötetään suoritettava tiedosto, operaatio (koulutus), aineiston sijainti, käytettävät painoarvot sekä iteraatiotuloksien nimi ja tallennusmuoto.

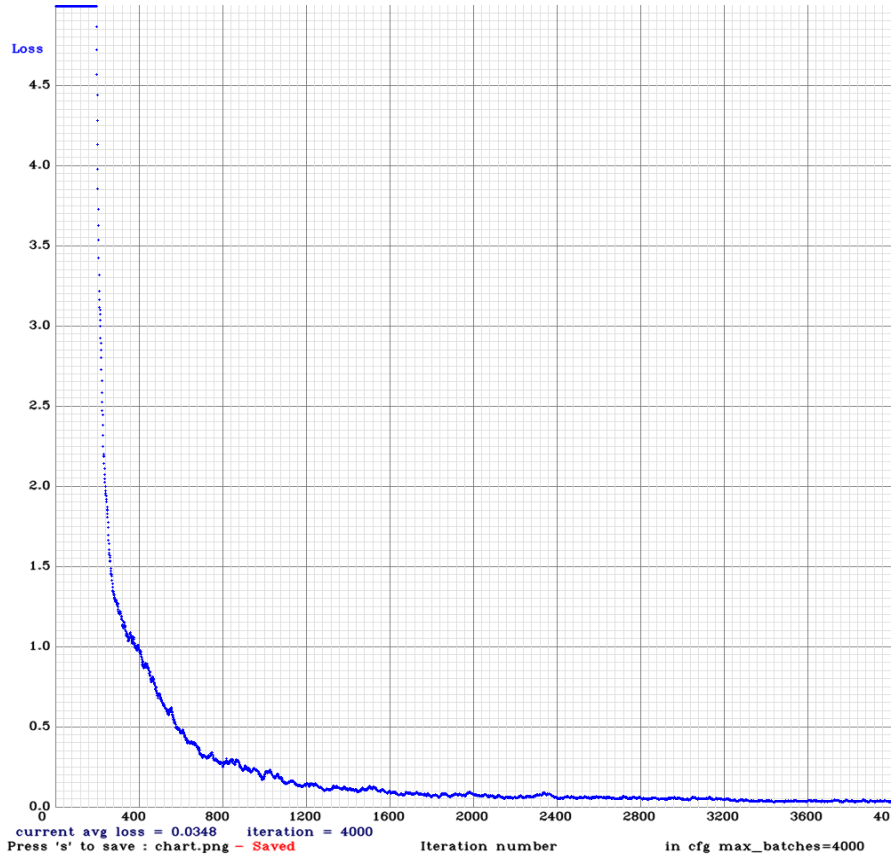
```
darknet.exe detector train .\data\nls.data .\cfg\YOLOv3-nls.cfg ..\..\..\..\darknet53.conv.74 >.\nls_trai-
ninglogs.txt 2>&1
```

Tämän jälkeen ohjelma näyttää ensin eri konvoluutioverkon osat ja sen jälkeen alkaa kouluttamaan algoritmia. Alla esimerkki iteraatiokierroksesta 1827, jossa ilmoitetaan esimerkiksi sen hetkinen tilanne koulutukseen käytetystä ajasta, tuloksista (esim. Avg loss kertoo tunnistuksen toimintakyvyn) ja käytettyjen kuvien määrän.

```
1827: 0.059710, 0.068989 avg loss, 0.001000 rate, 18.731000 seconds, 116928 images Loaded:
0.000000 seconds

v3 (mse loss, Normalizer: (iou: 0.750000, cls: 1.000000) Region 106 Avg (IOU: 0.850603, GIOU:
0.848220), Class: 0.999682, Obj: 0.998382, No Obj: 0.000178, .5R: 1.000000, .75R: 1.000000,
count: 4, loss = 0.059027, class_loss = 0.017696, iou_loss = 0.041331
```

Yhteensä 4000 iteraatiokierroksen suorittamiseen kului 16,5 tuntia ja ohjelma tallensi painoarvotiedoston joka tuhannella iteraatiokierroksella. Tämän avulla voidaan koulutus aloittaa aiemmasta kohdasta, jos jokin menisi koulutuksessa vikaan. Koska koulutus sujui hyvin, tämä ei ollut tarpeellista, ja painoarvotiedostoja kertyi yhteensä viisi kappaletta. Koulutuksen tulosta voidaan seurata ohjelman synnyttämästä graafista, joka on kuvattu kuvassa 26. Koulutus voidaan Darknet-dokumentoinnin perusteella lopettaa, kun Avg. loss on saavuttanut arvon 0,60730 (Darknet 2020).



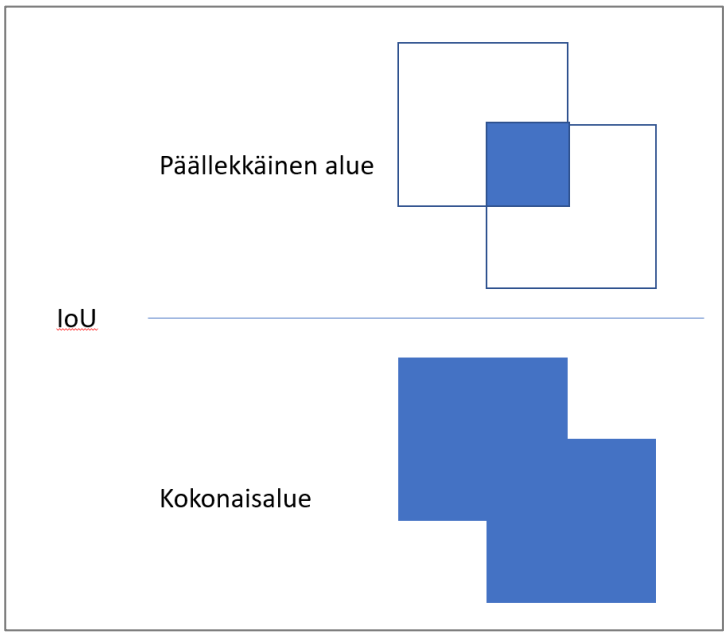

*Kuva 26. Kuvaaja oppimisprosessista. Y-akselilla näkyy loss-arvo, jonka pitäisi olla mahdollisimman lähellä nollaa, ja X-akselilla iteraatiokierros.*

## 4.3 Tunnistustarkkuuden arviointikeinot

Liu et al. (2019) kertovat algoritmien toimintakyvyn mittaamisen pohjautuvan yleensä kolmeen kriteeriin. Nämä ovat nopeus, jota mitataan kuvataajuudella (fps, *frames per second*), tarkkuus (*precision*) ja tarkkuusmuistutus (*recall).* Yleinen on myös AP (*Average precision),* joka lasketaan yleensä yksittäiselle kohdeluokalle, sekä mAP (m*ean Average Precision*), joka lasketaan, mikäli tunnistuksessa on mukana useampi kuin yksi luokka. Lisäksi tavallisesti käytetään leikkauksen ja unionin osamäärää (IoU = *Intersection over Union*) mittaria. Tähän viitataan tässä työssä myöhemmin IoU:na. Myös Li et al. (2019) keskustelevat artikkelissaan yleisimpien kohteentunnistusalgoritmien kouluttamisesta Pano-RSOD-aineistolla ja evaluoivat näiden antamia tuloksia mittarein AP (*Average precision*) ja mAP *(Mean average precision*).

Tässä työssä käytetään arviointikeinoina mittareita kuvataajuus (*FPS*), *Intersection over Union* (IoU), *precision* ja *recall* niiden yksinkertaisuuden ja käytettävyyden vuoksi. Kuvataajuus lasketaan katsomalla, kuinka kauan algoritmilla kestää tehdä tunnistus yhdeltä kuvalta. IoU:ssa (kuva 27) lasketaan, miten tarkastellaan käsitellyn ja annotoidun aineiston (*ground truth*) suorakulmarajaukset (*bounding*

*box*) vastaavat algoritmin tekemiä ennustettuja (*predicted*) suorakulmarajauksia. IoU:ssa lasketaan alue, jonka nämä ovat päällekkäin jaettuna rajauksien kokonaisalueella. Mikäli nämä rajaukset osuvat yhteen, on kyseessä oikea positiivinen (*true positive*) havainto. Jos useampi ennustettu rajaus osuu samaan, valitaan näistä parhaan IoU:n lopputulos oikeaksi positiiviseksi ja muut rajaukset vääriksi positiivisiksi (*false positive*).




*Kuva 27. Mukailtu kuvaus IoU:n laskemisesta (Rosebrock 2016) mukaan.*

Jos ennusteet eivät täsmää testausaineistossa havaittujen kohteiden kanssa, on kyseessä väärä negatiivinen (*false negative*), ja mikäli ennusteita tapahtuu ilman koulutusaineistoa, on kyseessä jälleen väärä positiivinen (false positive) (Li et al. 2019). IoU:n arvo voi vaihdella 0-1 välillä, joskin yleensä positiiviseksi havainnoksi määritellään yli 0,5 IoU-arvon saaneet havainnot (Liu et al. 2019, Stenroos 2017, 38). Tämä tulos kuitenkin vaihtelee ja joissain tutkimuksissa vaaditaan korkeampi tulos (Liu et al. 2019).

*Precision* eli tarkkuus lasketaan jakamalla oikeat havainnot väärien ja oikeiden havaintojen summalla. Sillä viitataan siihen, kuinka tarkasti mallinnus tuottaa relevantteja tuloksia (Garg et al. 2019). *Recall* lasketaan jakamalla oikeat havainnot kaikkien mahdollisten havaintojen kanssa. Se kertoo, kuinka hyvin malli onnistuu löytämään kaikki oikeat havainnot kokonaisesta havaintojoukosta (Garg et al. 2019).

## 4.4 Sijainnin määrittäminen kuvilta

Laskennassa pyritään selvittämään kärkikolmioiden sijainnit pohjautuen YOLO:n antamien tunnistusten *bounding box* -rajauksiin sekä kuvanottopaikkojen koordinaatteihin. Lisäksi tarvitaan tieto kameran suunnasta sekä laskennan parantamiseksi useampia kuvia samasta kolmiosta. Laskentaa käytetty skripti löytyy liitteen 3 osiosta ”XML_to_MAP.ipynb”. Skriptin hyödyntämiseen tarvitaan ArcGIS Pro 2.5-ohjelmistoa. Testauksessa tunnistettujen kärkikolmioiden sijaintien laskeminen toteutettiin Esrin Tommi Terävän toimesta.

Työvaiheet visuaalisesti kuvattuna pohjautuen Terävän (Terävä 2020) kanssa käytyyn keskusteluun:

1. Alemmassa kuvassa (kuva 28) näkyy havaintojen perusteella muodostetut kuvapisteet ilmakuvassa. Kauempaa havaitut kohteet tuottavat yleisesti enemmän virhettä sijaintiin kuin läheltä otetut. Kuvalta laskennan algoritmiin lisättiin logaritminen muuttuja, joka korjaa virhettä pidemmiltä matkoilta.

*Kuva 28. Kuvassa näkyvät pisteet muodostettiin havaintojen perusteella.*

2. Eri karttapistejoukoista muodostetaan klusterit, joilla ne erotellaan toisistaan. Tähän käytettiin ”*Density-based Clustering”*-työkalua. Kuvassa 29 näkyy eri sävyillä mainitut karttapiste-joukot.

*Kuva 29. Samoista kärkikolmioista tehdyt havainnot klustereina.*

3. Klustereiden keskipisteen avulla voidaan arvioida kärkikolmion sijainti. Tähän käytettiin ”*Mean Center*”-työkalua. Tulokset kahden kärkikolmion sijaintiin on esitetty kuvassa 30.

*Kuva 30. Kuvassa näkyvät pisteet muodostettiin aiemmin mainittujen klustereiden keskipisteistä.*

# 5. Tulokset

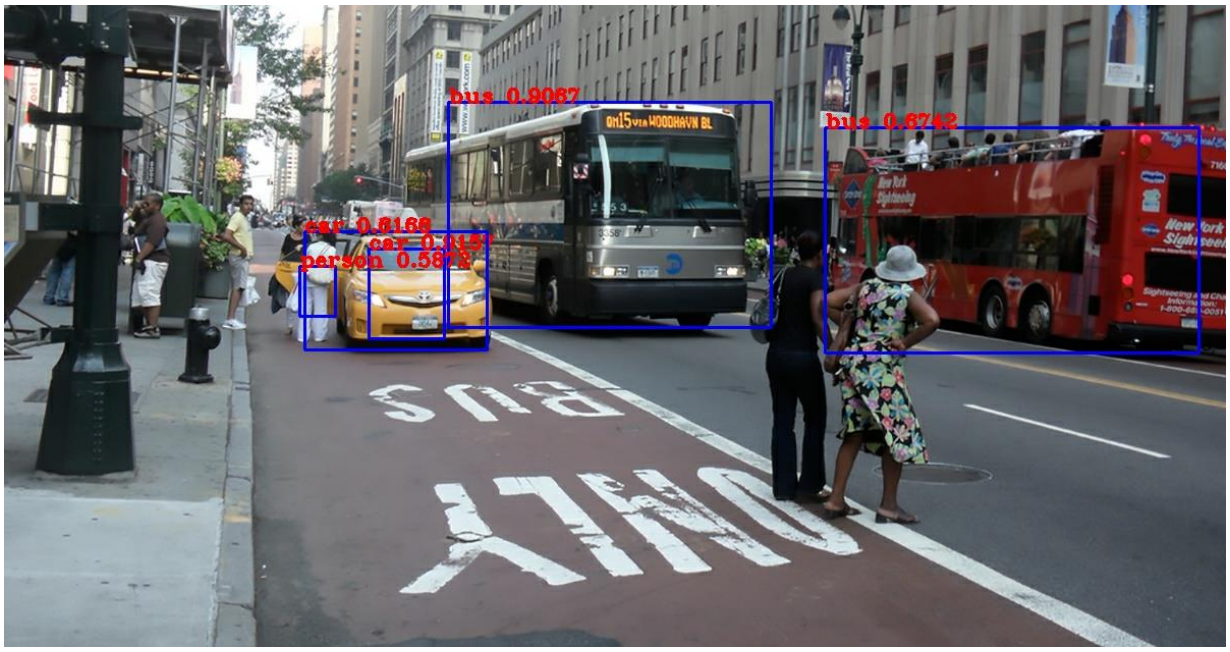
## 5.1 YOLO:n hyödyntäminen omassa työssä

Tässä työssä tutkitaan YOLOv3- kohteentunnistusalgoritmin käyttöä omilla kuva-aineistoilla. Työssä testataan tunnistamista staattisilta panoraamakuvilta sekä niistä pilkotuilla kuvilla. Koulutusaineistona käytetään mainittuja Mapillaryn kuva-aineistoja sekä muita kuin testauksessa käytettyjä panoraamakuvia. Alla olevissa kuvissa (kuva 31 ja 32) on esitetty, miten oletuspainoarvotiedostoa (*weights file*) hyödyntäen onnistutaan tunnistamaan valittuja oletuskohteita kuvilta. Tämän painoarvotiedoston voi ladata suoraan verkosta ja se on valmiiksi koulutettu tunnistamaan esimerkiksi autoja, busseja ja ihmisiä. Kuten kuvasta 31 huomataan, ei algoritmin toiminta ole kuitenkaan kovin luotettavaa Mapillary:n suuripikseliseltä kuvalta. Kuvalta onnistutaan tunnistamaan vain keskellä oleva auto, vaikka kuva sisältää useampiakin autoja. Myöskään tunnistuksen rajaus ei ole kovin hyvä, sillä suuri osa autosta jää tehdyn *bounding box* -rajauksen ulkopuolelle.

Kuva 32 toimitetaan osana Darknet-tietolähdettä (*repository)*, jotta algoritmin toimintaa voidaan tes-

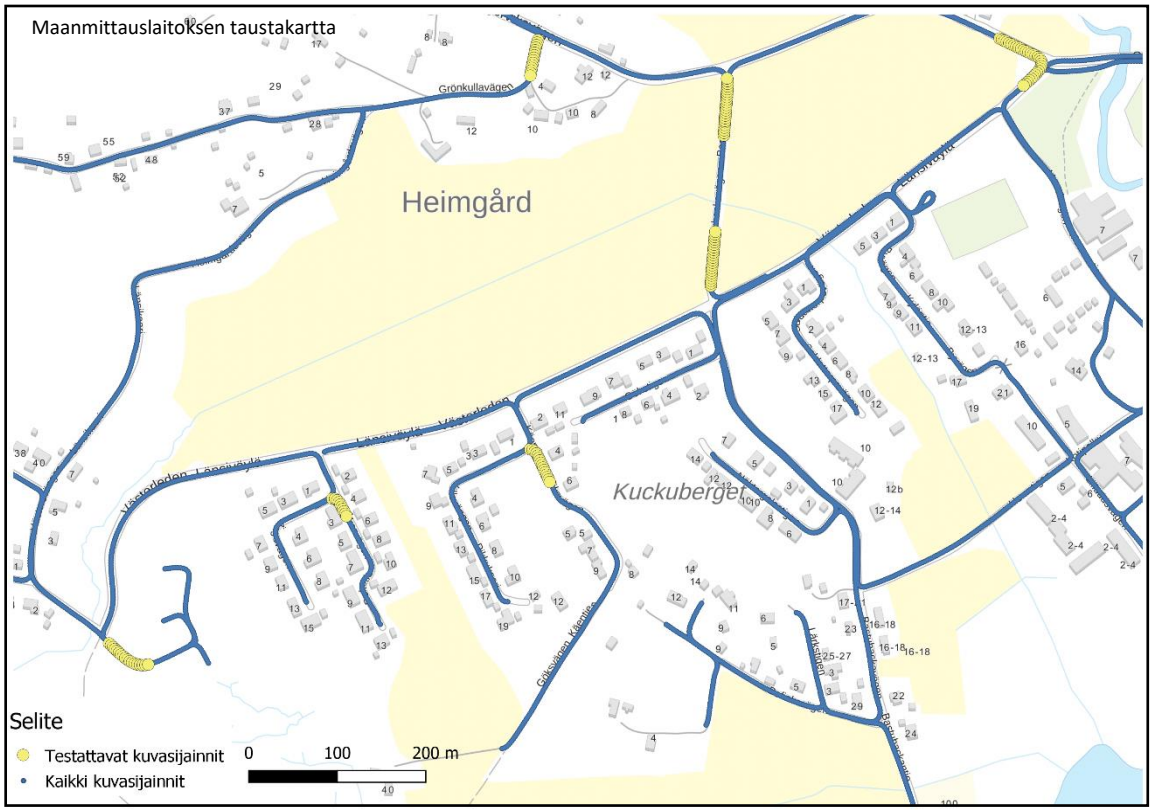

*Kuva 31. Tunnistus Mapillaryn panoraamakuvalta ilman koulutusaineistoja. Kuten kuvasta näkyy, ei YOLO toimi optimaalisesti korkean pikseliarvon (1920 x 1080) kuvilta ja se tunnistaa vain yhden auton, vaikka kuvassa on näkyvillä useampikin kohde.*

tata. Siitäkin voidaan havaita, ettei tunnistustarkkuus ole kovin luotettava. Esimerkiksi useampi ihminen ja ajoneuvo jää tunnistamatta tältä kuvalta.




*Kuva 32. Darknet-verkon testikuvasta oletuspainoilla koulutettu ohjelma tunnistaa valmiiksi koulutettuja kohteita, kuten tässä tapauksessa auton, ihmisen ja bussin.*

Kun koneoppimisalgoritmi saadaan koulutettua, on seuraavana vaiheena testata sen toimivuutta uudella aineistolla, jota ei ole annotoitu. Tähän valikoitui yhdeltä Inkoon ajoreitiltä osakokonaisuus, jossa oli yhteensä 144 kärkikolmion sisältävää kuvaa (kuva 33).

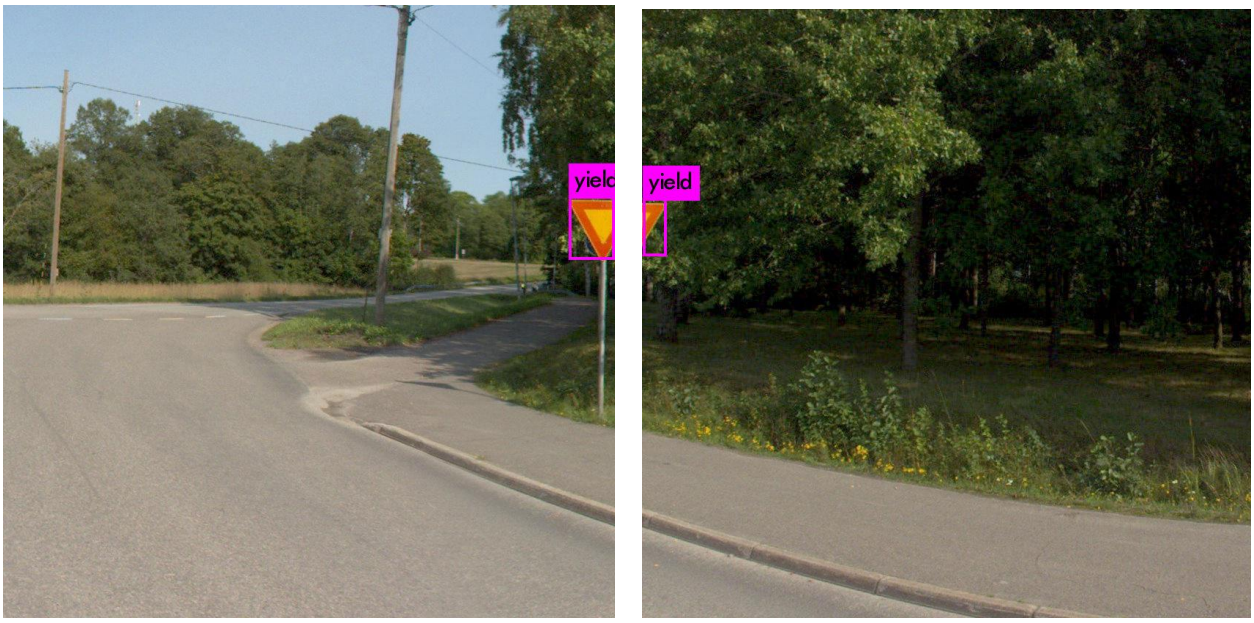



*Kuva 33. Inkoon kaikki kuvasijainnit ja testaukseen valitut kuvauspaikat*

Näistä tehtiin jaottelu osakokonaisuuksiin (7200 kuvaa) patches.py:n avulla. Testaus tapahtui melkein samalla komennolla kuin koulutuksen aloittaminen, jossa ensin viitataan suoritettavaan ohjelmaan ja sitten testauskomentoon, datatiedostoon ja konfiguraatiotiedostoon, jotka ovat samoja kuin koulutuksessa. Näiden lisäksi viitataan koulutuksessa rakennettuun painoarvotiedostoon ja määritetään pienin mahdollinen kynnysarvo (*threshold value)*, jolla tunnistus tapahtuu. Omassa testauksessa tämä määritettiin *-thresh* 0.8, jolla viitataan, että luottamustason pitää olla vähintään 0.8, jotta tunnistus hyväksytään. Oletuksena tämä on 0.25, jonka pohjalta tulee paljon vääriä havaintoja, mikä havaittiin myös omassa aineistossa. Esimerkkejä näistä tapauksista käydään läpi työn keskusteluosiossa. Lisäksi komennossa määritetään tulostiedosto, valitaan, että ei näytetä väliennustuksia (nopeuttaa laskentaa) ja määritellään tiedostopolku testattaviin kuviin sekä tallennettava tulostiedosto.

```
darknet.exe detector test .\data\nls.data .\cfg\YOLOv3-nls.cfg
.\data\backup_20022020\YOLOv3-nls_final.weights -thresh 0.8 -ext_output -dont_show <
data\”tiedostopolku kuviin”.txt > ”tulostiedoston nimi”.txt
```

Testaukseen valituista kuvista löytyi kärkikolmio yhteensä 137 kohteesta, joista duplikaattien osuus oli kolme, jotka poistettiin lopullisesta tuloksesta. Duplikaatteja syntyi jakamisen perusteella ja duplikaateista valittiin korkeamman luottamustason saanut havainto, josta esimerkki kuvassa 34. Vasemman puolisessa kuvassa luottamustaso oli 99 % ja oikeanpuolimmaisen 98%, jolloin ensin mainittu otettiin lopulliseen tulokseen. YOLO tuottaa purppuran värisen tunnistusrajauksen kuville sekä luokan, joka on nähtävissä kuvalla.

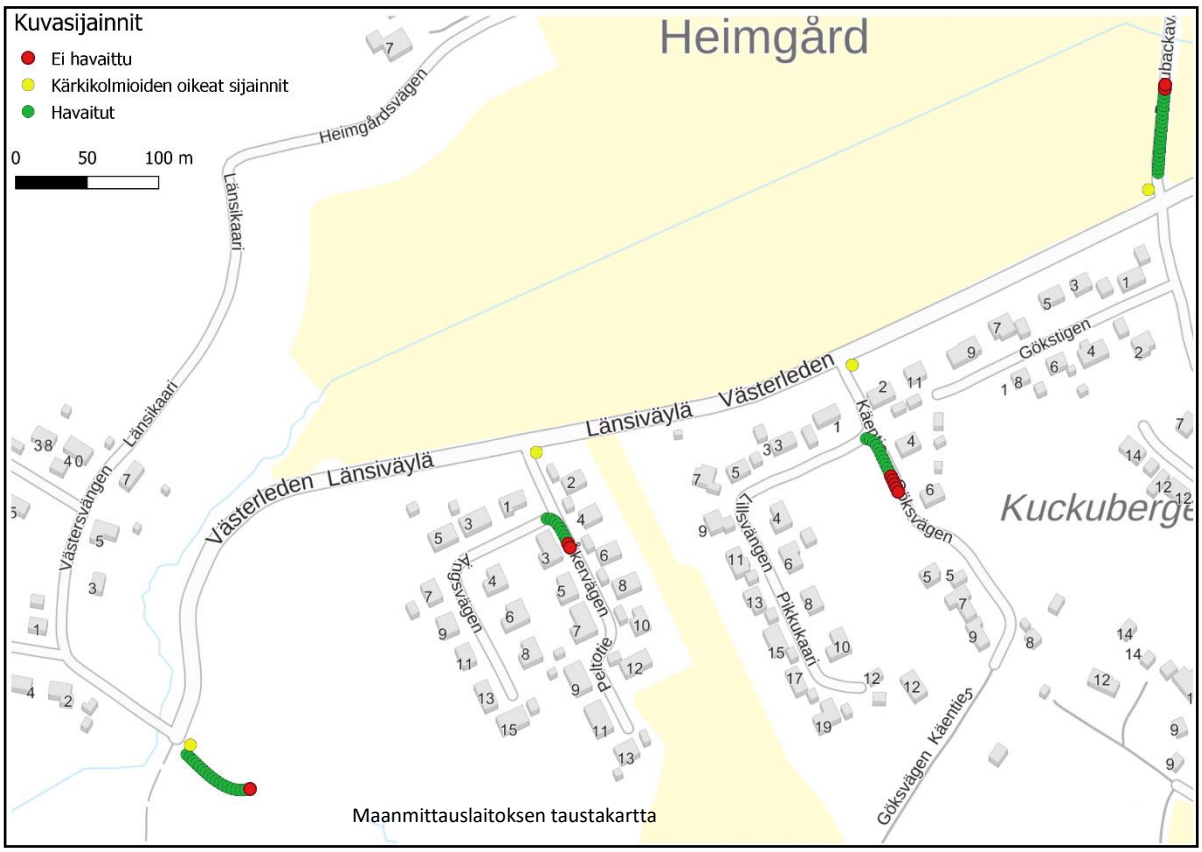



*Kuva 34. Peräkkäin otetut kuvat, joista oikeanpuoleinen sai korkeamman luottamustason.*

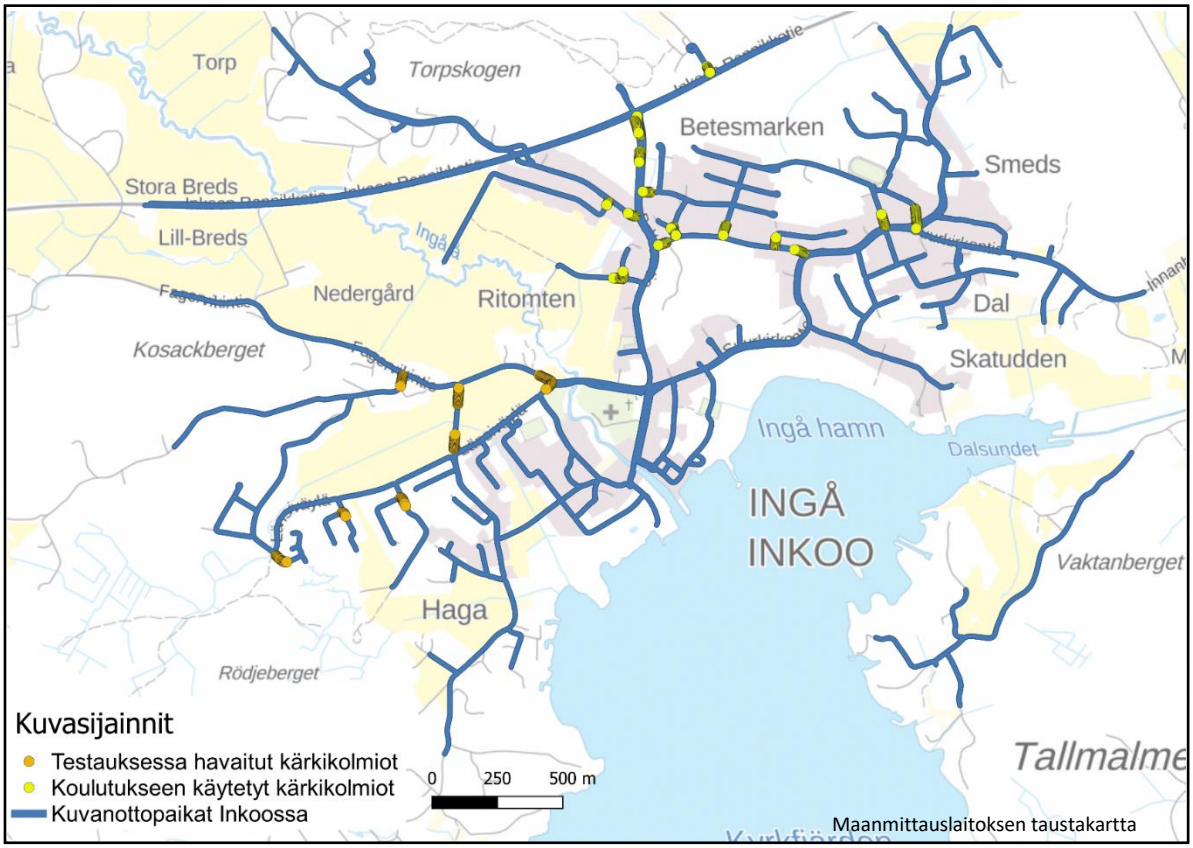
Tämän myötä löytyi yhteensä 134 kuvaa, joissa oli havaittu kärkikolmio yli 80 prosentin luottamuksella. Havaitsematta jäi siis kymmenen kärkikolmiota, joista suurin osa on kuvattuna varsin etäältä itse kohteesta. Kuvassa 35 on esiteltynä kaksi tämänlaista kuvaa, joista vasemman puolisessa näkymässä kärkikolmio on hyvin pienenä näkyvissä. Oikeanpuoleinen ei taas ole kovin kaukana, mutta mahdollisesti varjoisuus yhdistettynä etäisyyteen vaikutti sen tunnistamiseen.

*Kuva 35. Esimerkkikuvat, joissa kohteita ei havaittu. Vasemmanpuoleisessa kuvassa kärkikolmiota ei havaittu (luottamustaso alle 25 %). Oikeanpuoleisessa kuvassa kärkikolmio havaittiin vain 70 % todennäköisyydellä, minkä vuoksi sitä ei oteta lopullisessa tarkastelussa silti huomioon.*

Kuvassa 36 on nähtävissä vihreällä ne kuvasijainnit, joissa kärkikolmiot havaittiin ja punaisella ne, joita ei havaittu. Kärkikolmioiden oikeat sijainnit näytetään keltaisella. Kuten kartasta näkyy, on kaikki kymmenen tunnistamatonta kuvasijaintia kaukana kärkikolmiosta. Algoritmin toiminta on ymmärrettävästi sitä heikompaa, mitä pienempänä kohde on kuvalla nähtävissä. Kuvassa 37 esitetään kartta, johon on koostettu testauksessa ja koulutuksessa käytetyt kuvasijainnit sekä kaikki kuvanottopaikat Inkoossa. Kuten kuvasta 37 näkee, on koulutus- ja testausaineisto kerätty eri paikoista, jotta nämä eivät sisällä samoja kuvauspaikkoja. Tämän myötä vältyttiin ylisovittamiselta.

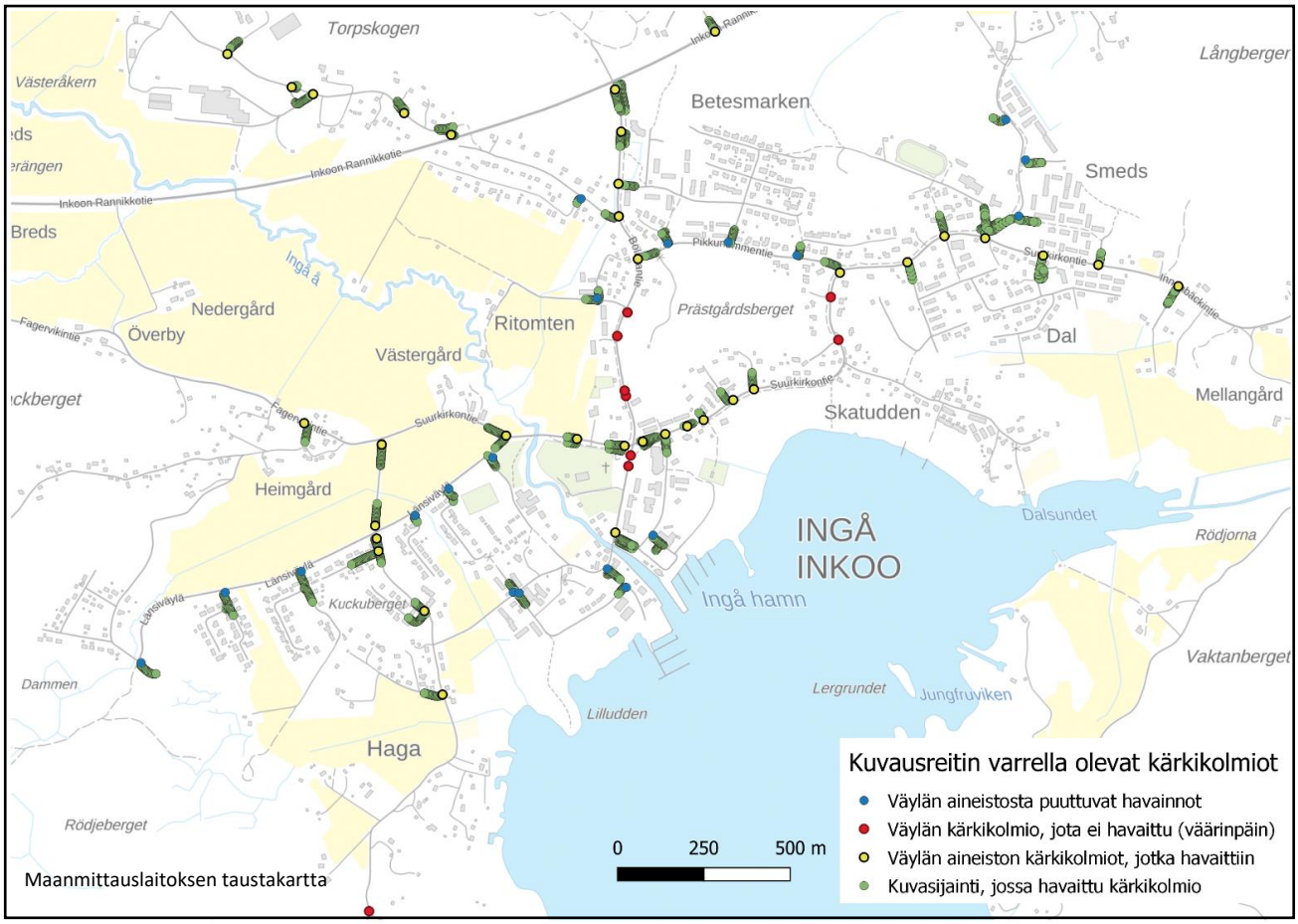



*Kuva 36. Kuvassa nähtävillä kuvasijainnit, joissa kärkikolmiot joko havaittiin tai ei havaittu sekä kärkikolmioiden oikeat sijainnit.*



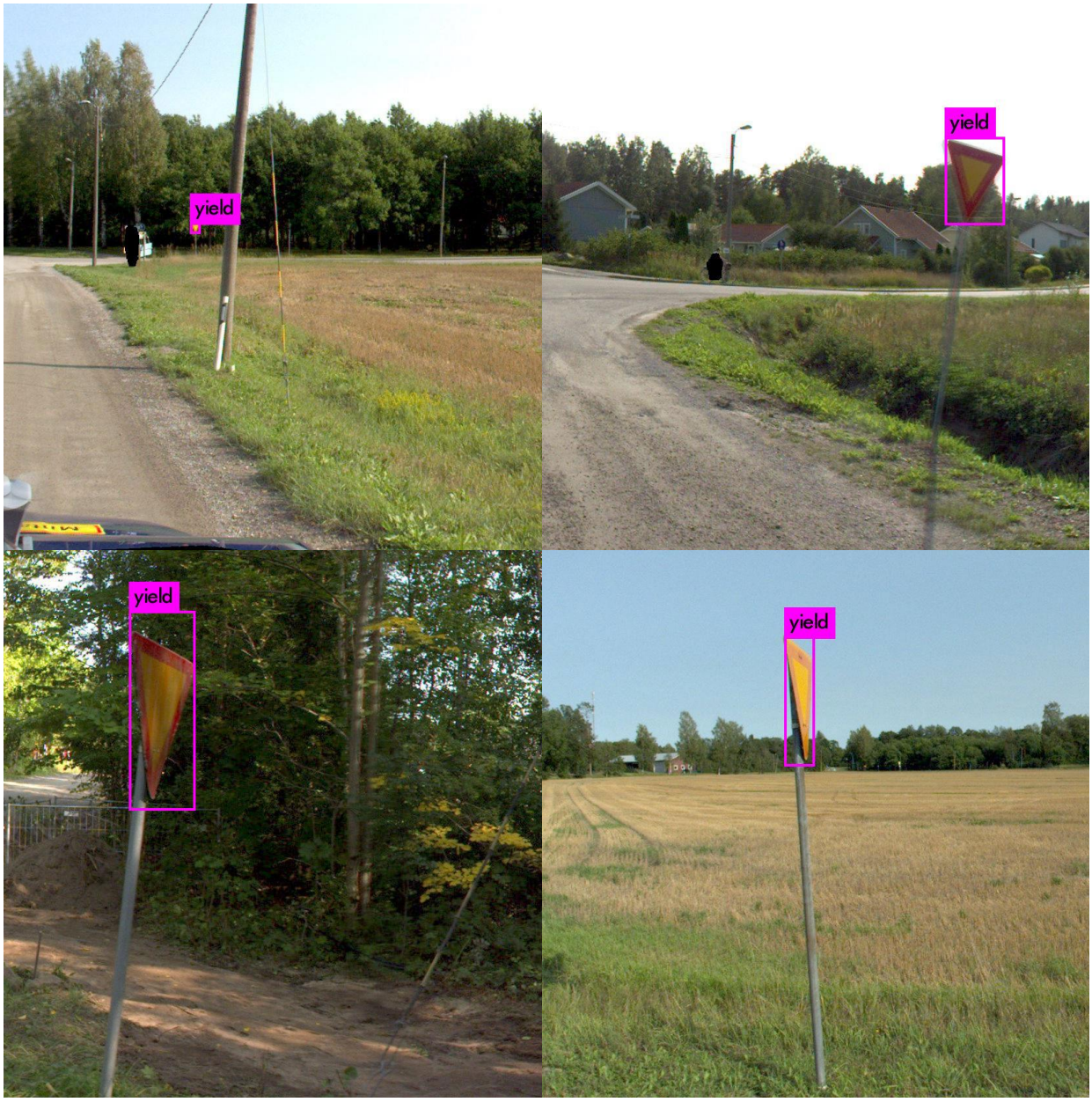

*Kuva 37. Tiivistelmä kaikista kuvasijainneista sekä testaukseen ja koulutukseen käytetyistä sijainneista.*

Väyläviraston aineistosta puuttui 19 kärkikolmiota, jotka havaittiin tämän tutkimuksen pohjalta. Kuvassa 38 on merkitty sinisellä nämä 19 kärkikolmiota. Kuvassa näkyy punaisena pisteet, joissa on Väyläviraston aineiston mukaan kärkikolmio, mutta sitä ei havaittu. Tämä johtui siitä, että kuvausauto ajoi kärkikolmion väärältä puolelta, jolloin kolmio näyttäytyi väärinpäin. Keltaisella symbolilla kartalle on merkitty havainnot, jotka ovat yhtenevät Väyläviraston aineiston kanssa. Vihreänä kartalla näkyvät kuvasijainnit, joista kärkikolmio on havaittu manuaalisessa tarkastelussa.




*Kuva 38. Kuvausreitiltä tunnistetut kärkikolmiot verrattuna Väyläviraston aineistoihin.*

Kuvassa 39 esitetään luottamustason 100 % tunnistuksia, joista on nähtävillä algoritmin toimintakyky haastavissakin tunnistuksissa. Ensimmäinen kuva on otettu varsin kaukaa, toinen on tärähtänyt ja kolmas sekä neljäs kuva on otettu vaikeasta kulmasta. Kuvissa näkyy myös algoritmin kärkikolmiolle ennustama koko, joka ei kuitenkaan kolmannessa kuvassa ole erityisen tarkka.




*Kuva 39. Esimerkkitunnistuksia 100 % luottamustasolla.*

## 5.2 Pilkottujen panoraamakuvien tulosten tarkempi analyysi

Havaintoja oli kaikkiaan 144, joista onnistuttiin tunnistamaan 134. Tässä työssä arvioidaan yli 0,5 IoU-arvot oikeiksi positiivisiksi (*true positive*), alle 0,5 IoU-arvot vääriksi positiivisiksi (false positive), ja ei-toteutuneet ennustukset vääriksi negatiiviksi (10 kpl).

*Taulukko 1. Pilkotuilta kuvilta toteutetun tunnistuksen tunnusluvut.*

| | |
|---|---|
| Määrä | 144 |
| Tunnistukset | 134 |
| True positive (IoU > 0,5) | 133 |
| False positive(IoU < 0,5) | 1 |
| False negative | 10 |

*Precision* eli tarkkuus lasketaan jakamalla oikeat havainnot väärien ja oikeiden havaintojen summalla. Sillä viitataan kuinka tarkasti mallinnus tuottaa relevantteja tuloksia (Garg et al. 2019) eli kuinka moni positiivista havainnoista oli oikeita:

$$Precision = \frac{True\ positive}{True\ positive\ + False\ positive}$$

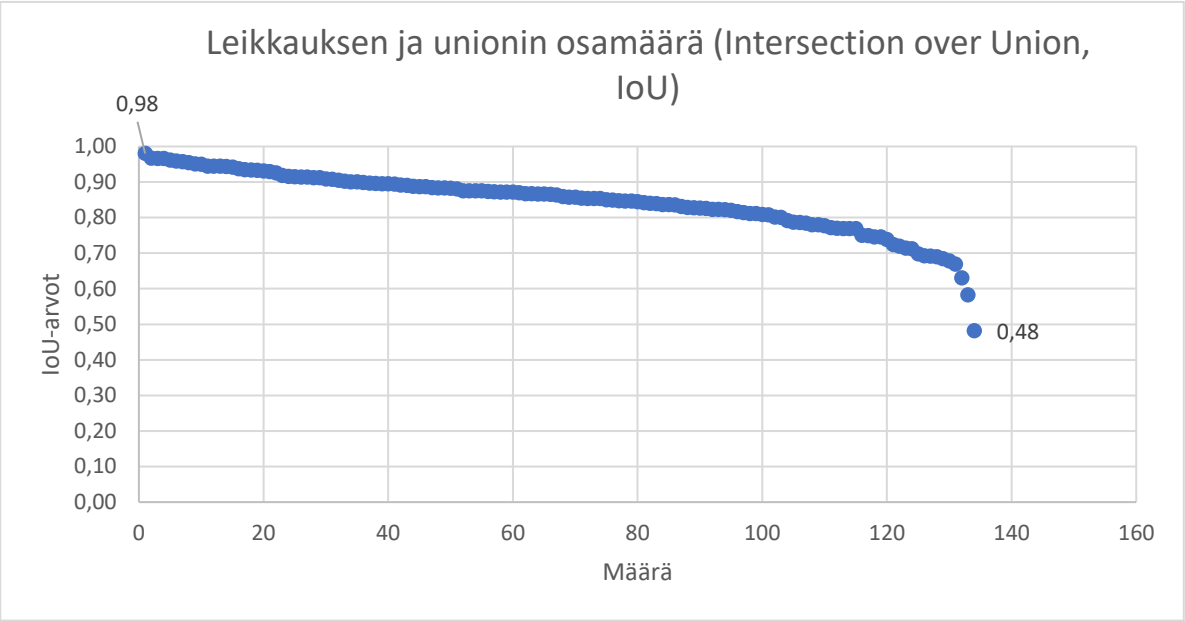
$$0.99253731343 = \frac{133}{133\ + 1}$$

*Recall* lasketaan jakamalla oikeat havainnot kaikkien mahdollisten havaintojen kanssa. Se kertoo, kuinka hyvin malli onnistuu löytämään kaikki oikeat havainnot kokonaisesta havaintojoukosta (Garg et al. 2019) eli kuinka suuri osuus oikeista positiivista havainnoista tunnistettiin oikein:

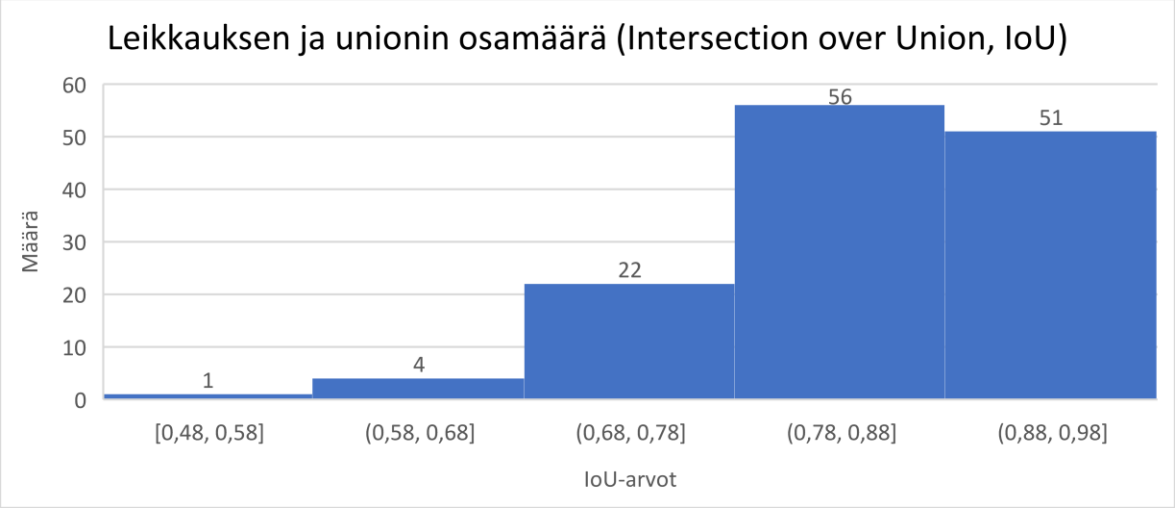
$$Recall = \frac{True\ positive}{True\ positive\ + False\ negative}$$

$$0.93006993007 = \frac{133}{133\ + 10}$$

Metriikan laskemiseen käytettiin IOU_function.py-koodia (liite 2), jossa määritellään YOLO:n antamat *bounding box* -rajaukset tunnistuksista sekä itse tehdyt annotaatiot ja vertaillaan näiden yhteyttä. Yhteensä havaintoja oli 134, joiden IoU-arvot esitetään kuvissa 40 ja 41. IoU-arvojen tunnusluvut puolestaan esitetään taulukossa 2. Melkein kaikki havainnot sijoittuvat yli 0,5 arvon ja yli sata havaintoa oli arvon 0,8 yläpuolella. Keskiarvo havainnolle oli 0,86 ja mediaani 0,85. Näiden perusteella voidaan arvioida algoritmin suoriutuneen hyvin tunnistuksessa sekä sijaintien selvittämisessä. Kuvataajuuden vaihtelun tunnusluvut esitetään taulukossa 3. Yksi kuva tarkistettiin keskimäärin 66,3 millisekunnissa ja fps-arvo oli noin 15 kuvaa sekunnissa.




*Kuva 40. IoU-arvojen jakauma hajontakuviossa. X-akseli kuvaa havaintojen määrää ja Y-akseli IoU-arvoja.*




*Kuva 41. IoU-arvojen jakauma histogrammissa, josta voi tarkistaa IoU-arvojen määrän eri asteikoilla.*

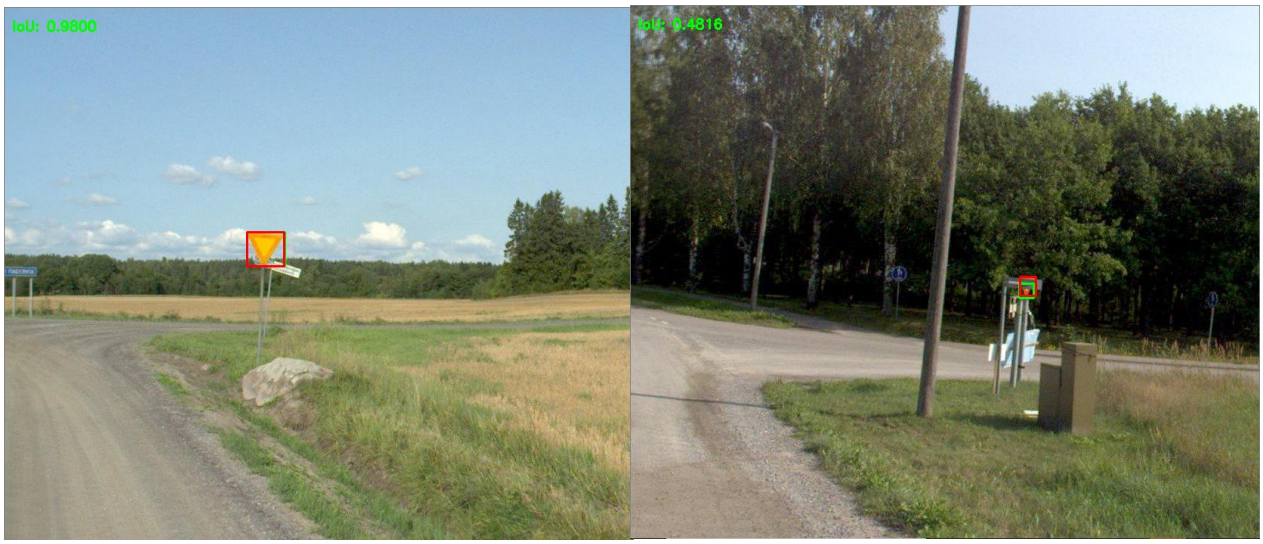
*Taulukko 2. IoU-arvojen tunnusluvut.*

| | |
|---|---|
| Määrä | 134 |
| Min | 0,48 |
| Max | 0,98 |
| Keskiarvo | 0,86 |
| Mediaani | 0,85 |
| Moodi | 0,94 |
| Keskihajonta | 0,08 |

*Taulukko 3. Pienten kuvien kuvataajuus (frames per second) ja muut nopeuden tunnusluvut yksittäisistä kuvista millisekunteina*

| | |
|---|---|
| Keskiarvo | 66,25 |
| Mediaani | 65,13 |
| Max | 142,45 |
| Min | 61,82 |
| FPS (1000/keskiarvo) | 15,09 |

Alempana olevissa kuvissa (kuva 42) on koottu paras ja huonoin IoU-arvo pilkotuilta kuvilta tunnistuksessa. Vasemmanpuoleisessa kuvassa IoU-arvo on erittäin hyvä 0,98 ja oikeanpuoleisessa kuvassa varsin huono 0,48. On kuitenkin hyvä huomata jälkimmäisen kuvan olosuhteiden olleen varsin epäedulliset, sillä suuri osa kärkikolmiosta on osoitekyltin peitossa. Kuvassa 43 taas on esimerkki kaukaa otetusta kuvasta, joka on silti IoU-arvoltaan hyvä (0,86).

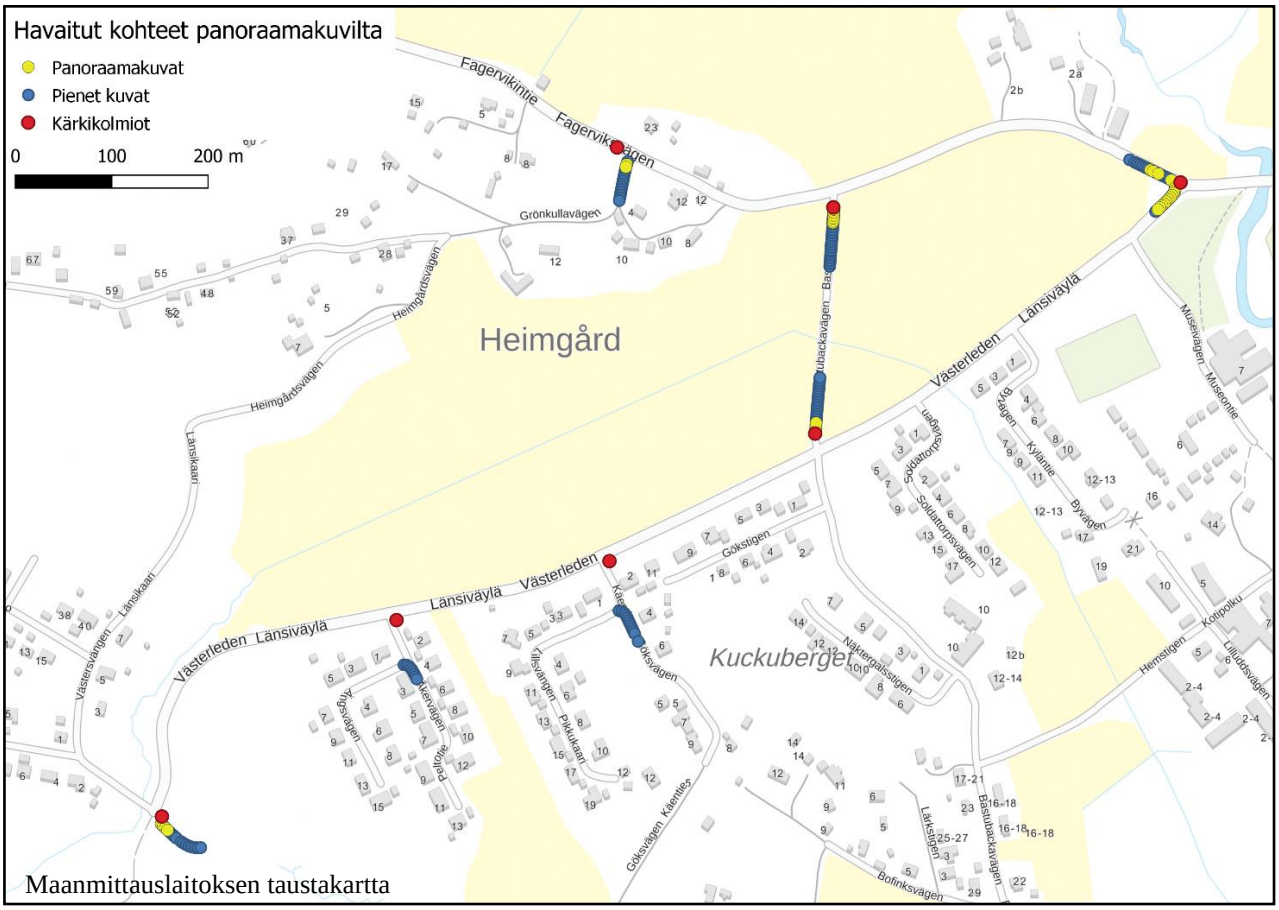



*Kuvat 42. Vasemmanpuoleisessa kuvassa parhaan IoU-arvon kuva ja oikeanpuoleisessa huonoin IoU-arvo.*


*Kuva 43. Kaukaa otettu kuva, jonka IoU-arvo 0,86.*

## 5.3 Vertailu panoraamakuvien ja niistä pilkottujen kuvien välillä

Panoraamakuvat ajettiin luodun YOLO-verkoston läpi, jotta voidaan verrata osuuko YOLO:n tunnistukset yhteen panoraamakuvien ja pilkottujen osakokonaisuuksien välillä. Panoraamakuville asetettiin tunnistuksen kynnysarvoksi myös 80 %, jotta tunnistukset olisivat tarkkoja. Verkosto ajettiin samoille panoraamakuville, joista aiemmin käytetyt pilkotut kuvat muodostuivat (144 kpl), jotta tulokset ovat vertailukelpoisia. Pienemmät osakokonaisuudet oli siis pilkottu näistä vertailtavista panoraamakuvista. Tuloksia voi tarkastella taulukosta 4, jossa näkyy tunnistuksen onnistuneen 35 kohteelle ja epäonnistuneen 101 kohteelle. Lisäksi vääriä positiivisia tuli yhteensä 10 kappaletta. Kahdelta kuvalta tuli sekä väärä positiivinen että oikea tunnistus, minkä takia tunnusarvoja tuli enemmän kuin 134. Alempana näkyvästä kartasta (kuva 44) myös nähdään, kuinka etäisyys vaikuttaa tunnistukseen panoraamakuvilta. Mitä lähempänä kärkikolmio on, sitä tarkemmin tunnistuksessa panoraamakuvalta onnistutaan. Kuvan 44 kahdelta reittiosuudelta ei onnistuttu havaitsemaan kärkikolmiota panoraamakuvilta.




*Kuva 44. Kartalla esitetään kuvanottopaikat, joissa kärkikolmio on nähty pilkotuilta kuvilta (pieniltä kuvilta) ja panoraamakuvilta sekä kärkikolmioiden sijainti.*

*Taulukko 4. Taulukossa koottuna panoraamakuvilta tunnistuksen tulos ilman IoU-luokitusta.*

| | |
|---|---|
| Oikea tunnistus (t*rue positive*) | 35 |
| Ei tunnistanut (f*alse negative*) | 101 |
| Väärä tunnistus (f*alse positive*) | 10 |
| Sisälsi oikean ja väärän tunnistuksen (TP & FP, lisätty sarakkeisiin) | 2 |
| Tunnisti kärkikolmion sisältävältä kuvalta väärän kohteen (FP sarakkeessa) | 8 |
| Precision | 0,78 |
| Recall | 0,26 |

Taulukon 4 arvoja voidaan verrata aiempiin pilkotuilta kuvilta tehtyihin havaintoihin, joissa tunnistustarkkuus oli huomattavasti korkeampi (*true positive* 133) sekä vääriä havaintoja ei tullut. Jos otetaan samat kriteerit IoU:n suhteen (taulukko 5, IoU:n on oltava yli 0,5, jotta tunnistus luokitellaan oikeaksi) tulokset heikkenevät tästä entisestään. Tarkkuus- (*precision*) ja tarkkuusmuistutusarvot (*recall)* olivat pilkotuilla kuvilla huomattavasti korkeammat (*precision* 0,99 ja *recall* 0,93) kuin panoraamakuvilta (ilman IoU-arvoja 0,78 ja 0,26 ja IoU-arvojen mukaan 0,63 ja 0,18). Samoin IoU-arvot olivat parempia sekä niitä oli oikeiden positiivisten myötä enemmän. Seuraavissa taulukoissa (taulukko 5 ja taulukko 6) on kootusti tietoa panoraamakuvilta tehtyjen tunnistusten IoU-arvoista.

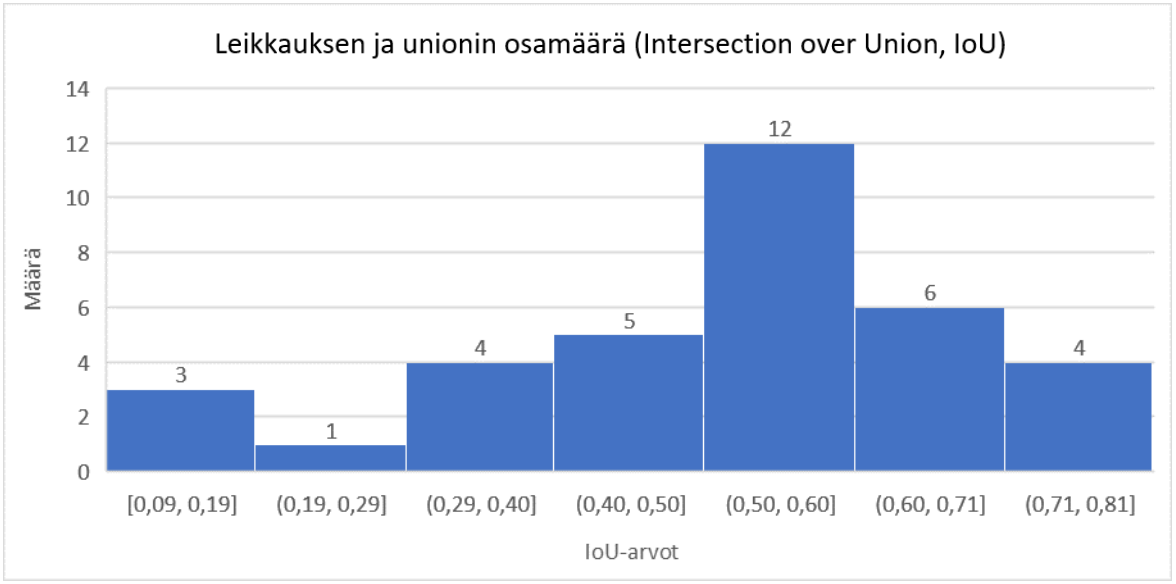
Kuvassa 45 on koottu histogrammiin panoraamakuvilta tehdyn tunnistuksen IoU-arvojen jakauma. Verrattuna pilkotuilta kuvilta tehtyyn histogrammiin (kuva 41) on arvot painottuneet 0,5–0,6 välille kun ne pilkotuilta kuvilta painottuivat 0,78–0,88 välille. Kuvassa 46 taas verrataan panoraamakuvien onnistuneita havaintoja samojen kuvien IoU-arvoihin pilkotuilta kuvilta. Taulukosta nähdään tunnistuksen onnistuneen paremmin pilkotuilta kuvilta sekä molempien tunnistusten huonoimmat ja parhaimmat IoU-arvot. Taulukossa 7 on kootusti tietoa panoraamakuvilta tehdyn tunnistuksen kuvataajuudesta (fps: 11), joka oli pienempi kuin pilkotuilta kuvista tehdyissä tunnistuksissa (fps: 15).

*Taulukko 5. Panoraamojen tunnistustarkkuus IoU-arvojen mukaan jaoteltuna.*

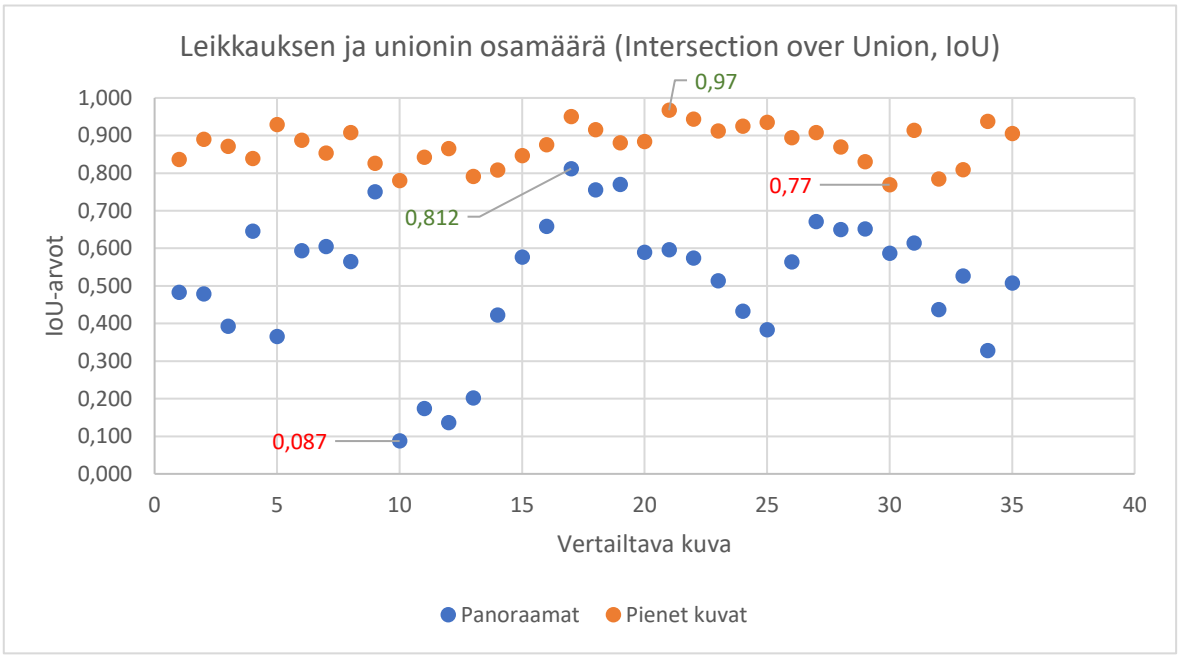
| | |
|---|---|
| Oikein (t*rue positive*) (IoU > 0,5) | 22 |
| Ei tunnistanut (f*alse negative*) | 101 |
| Väärä tunnistus (f*alse positive*) (IoU < 0,5) | 13 |
| Precision | 0,63 |
| Recall | 0,18 |

*Taulukko 6. Panoraamojen IoU-arvojen tunnusluvut.*

| | |
|---|---|
| Määrä | 35 |
| Min | 0,087 |
| Max | 0,812 |
| Keskiarvo | 0,52 |
| Mediaani | 0,56 |
| Keskihajonta | 0,18 |




*Kuva 45. Histogrammissa panoraamakuvien (35 kpl) IoU-arvot ja niiden jakautuminen.*




*Kuva 46. IoU-arvojen vertailu samoista kuvasijainneista pilkottujen kuvien (pienten kuvien) ja panoraamojen välillä.*

*Taulukko 7. Panoraamakuvien kuvataajuus (frames per second) ja muut nopeuden tunnusluvut yksittäisistä kuvista millisekunteina*

| | |
|---|---|
| Keskiarvo | 90,14 |
| Mediaani | 89,98 |
| Max | 180,52 |
| Min | 74,1 |
| FPS (1000/keskiarvo) | 11,09 |

Seuraavissa kuvissa (47 ja 48) on esimerkkejä tunnistuksen tarkkuudesta panoraamakuvilta. Kuvien oikealla puolella näytetään lähennetty kuva, jotta tulos olisi paremmin näkyvissä. Kuvassa 47 on parhaan IoU-arvon (0,81) saanut kuva, jossa kärkikolmio on kuvattu suhteellisen läheltä. Aiemman kartan (kuva 44) perusteella lähempää otetut kuvat tuottivat parempia tuloksia kärkikolmioiden tunnistuksessa. Alempana kuvassa 48 on taas kuvattuna huonoimman tuloksen saanut tunnistus 0,08, jossa kärkikolmio on erittäin pienenä kuvassa, minkä lisäksi se on hieman sivuttain.

*Kuva 47. Parhaan IoU-arvon saanut panoraamatunnistus (0,81) sekä lähennetty kuva kärkikolmiosta. (Vihreänä YOLO:n ennustus ja punaisella oikea rajaus ground truth)*

*Kuva 48. Huonoimman IoU-arvon saanut panoraamakuvatunnistus (0,08) sekä lähennetty kuva tunnistuksesta (vihreänä YOLO:n ennustus ja punaisella oikea rajaus ground truth).*

# 6. Keskustelu

## 6.1 Tutkimuskysymyksiin vastaaminen

Työssä selvitettiin panoraamakuvien ja niistä luotujen pilkottujen osakokonaisuuksien käyttöä kohteiden tunnistuksessa. Vertaillessa näiden eroja tunnistustarkkuudessa huomattiin algoritmin toimivan huomattavasti paremmin pilkotuilta kuvilta tunnistamisessa kuin vastaavista panoraamakuvista. Algoritmi toimi myös nopeammin sekä kohteiden sijainti kuvassa saatiin tarkemmin laskettua. Kohteita onnistettiin paikallistamaan myös panoraamakuvilta, mutta huomattava osa näistä jäi kuitenkin havaitsematta verrattuna pilkottuihin kuviin. Samalla vääriä tunnistuksia syntyi enemmän. Yleisesti kuvanottopaikan ja kärkikolmion välinen etäisyys vaikutti merkittävästi liikennemerkin havaitsemiseen. Toisin sanoen mitä suurempana kärkikolmio esiintyi kuvissa, sitä todennäköisemmin se myös havaittiin.

Tutkielmassa haluttiin myös selvittää kuinka hyvin YOLO:n tuottamien *bounding box* -rajausten avulla pystyy määrittämään kohteen oikean sijainnin. Tämä analyysi toteutettiin yhdessä Esri Finlandin Tommi Terävän kanssa. Kärkikolmioiden sijoittamisen tuloksia arvioitiin visuaalisesti vertaamalla tuotettuja klustereiden keskipisteitä ilmakuvilta pääteltäviin kärkikolmioiden sijainteihin. Keskipisteiden arvioitiin eroavan muutaman metrin oikeista sijainneista, mikä ei riitä tarpeeksi laadukkaaksi arvioksi sijainneista. Kuvilta laskennan antamia tuloksia olisi voitu arvioida tarkemmin selvittämällä maastossa kylttien oikeat sijainnit.

Väyläviraston avoimen aineiston vertailussa haluttiin selvittä, kuinka ajantasaista ja laadukasta tieto on sekä löytyykö YOLO:a hyödyntämällä uusia kärkikolmiohavaintoja Inkoosta. Tuloksena oli, ettei Väyläviraston aineistossa ollut merkittynä kaikkia alueen kärkikolmioita. On hyvä kuitenkin todeta, että merkkien havaitseminen on riippuvainen auton kuvausreitistä. Esimerkiksi useita Väyläviraston kärkikolmioita ei voitu todentaa, koska kärkikolmio oli väärinpäin kuvausautoon verrattuna.

## 6.2 Tutkimuksen arviointi

Työn hypoteesi oli, että tunnistus tulisi onnistumaan pilkotuilta kuvilta paremmin, koska algoritmien toiminta nojaa vahvasti syötettyjen kuvien kokoon. Mitä suuremman osan kohde täyttää kuvasta, sitä todennäköisemmin se myös havaitaan. Pienentämällä kuvakokoa entisestään olisi ollut mahdollista päästä ehkä vielä parempiin tuloksiin, mutta verrattuna muissa tutkimuksissa saatuihin (esim. Arcos-García et al. 2018) IoU, *precision* ja *recall* -arvoihin, algoritmi onnistui varsin hyvin. Vertailu eri aineistoihin on kuitenkin hyvin hankalaa, koska yleensä muissa tutkimuksissa on tutkittu algoritmin

toimintaa useammalla kuin yhdellä kohdeluokalla. Lisäksi ilman syvempää perehtymistä muissa tutkimuksissa käytettyihin aineistoihin on hankalaa arvioida luotettavasti ja tarkasti kuinka paljon vaikutusta erilaisella aineistoilla olisi ollut.

Omaa tutkimusta olisi voinut parantaa lisäämällä mukaan muitakin panoraamakuvilta näkyviä kohteita. Näitä olisi voineet olla esimerkiksi osoitekyltit, sähkötolpat, muuntajat, tieviitat, tiemerkinnät sekä muut liikennemerkit. Periaatteessa koulutusaineiston luonti olisi mahdollista siis mistä vaan kohteesta, kunhan siitä on riittävästi havaintoja. Ongelmana on lähinnä valittavien kohteiden määrän vaihtelu, niiden heterogeenisyys saman kohdeluokan sisällä sekä erotettavuus muista kuvan kohteista. Heterogeenisyyttä voi esiintyä esimerkiksi suurien kivien välillä. Niiden koko vaihtelee suuresti keskenään ja osassa voi olla vaikkapa maa-ainesta peitteenä. Tämä tarkoittaa, että koulutusaineistoa vaadittaisiin todennäköisesti paljon enemmän, jotta alisovittamista ei tapahdu.

Tunnistustarkkuutta olisi todennäköisesti voitu parantaa erilaisin apuvälinein, joita esiteltiin luvussa 2.5.2. Samoin algoritmia olisi voitu optimoida vieläkin nopeammaksi ja ehkä tarkemmaksi erilaisilla parametrimuutoksilla. Jos YOLO olisi koulutettu myös uudestaan pelkillä panoraamakuvilla, olisi tulokset voineet olla varsin erilaisia kuin nyt. Tähän ei kuitenkaan resurssien rajallisuuden vuoksi ryhdytty. Lisäksi koulutuksessa olisi voitu käyttää vielä haastavimpia kuvia, jotta erilaiset erikoistilanteet, kuten haastavat valoisuusolosuhteet tai kuvauskulman eroavaisuudet olisi otettu huomioon. Testauksessa onnistuttiin kuitenkin havaitsemaan myös haastavia kohteita, minkä vuoksi voidaan todeta, että ylisovittaminen onnistuttiin ottamaan hyvin huomioon koulutusaineiston luonnissa.

Ylisovittamista voisi tässä tapauksessa tapahtua, jos kuvat muistuttaisivat liikaa toisiaan ja sisältäisivät samanlaista informaatiota. Esimerkiksi Inkoossa suurin osa ajetuista reiteistä on keskustan ulkopuolella, jolloin maaseutumaiset tausta ovat yliedustettuina ja kaupunkimaiset taustat aliedustettuina. Panoraamakuvista luotiin opetusaineisto yhdeltä kuudesta ajoreitistä, missä kuvien ympäristö on enimmäkseen maanteitä ja tien vierustat metsän peittämiä. Ylisovittamista pyrittiin myös välttämään lisäämällä mukaan Mapillaryn kuva-aineistot, joissa oli enemmän hajontaa kuvanottopaikkojen välillä. Osa näistäkin kuvista sijoittui tiheästi asutuille kaupunkialueille, jotka eivät näin vastaa Inkoon kunnan alueita. Lisäksi osa kuvista oli otettu Ruotsista, jossa kärkikolmiot ovat hieman eri näköisiä (ohuempi reunaviiva) kuin Suomen alueella (kuva 49).


*Kuva 49. Ruotsin kärkikolmio vasemmalla puolella ja suomalainen oikealla puolella. Kuvat Mapillaryn aineistosta.*

Yhteensä kärkikolmiohavainnon mahdollistavia kuvauspisteitä kertyi 1287. Aineisto valittiin manuaalisesti, joten oli odotettavissa, että osa havainnoista voisi olla väärin valikoitu. Esimerkkejä valitun aineiston puutteellisuudesta on nähtävissä seuraavan sivun kuvassa 50. Ensimmäisenä olevasta kuvasta nähdään esimerkki, jossa automatiikka on Trimblen laitteessa anonymisoinut kärkikolmion väärin. Sitä seuraavassa kuvassa, joka on otettu hetkeä myöhemmin, kärkikolmio on taas näkyvissä. Kolmannesta kuvasta kärkikolmio on kuvattu takaa, jolloin sitä ei voida havaita. Neljännessä kuvassa kärkikolmio puolestaan näkyy pienenä oikeassa reunassa, mutta kuva on otettu liian kaukaa, jotta algoritmi voisi luokitella sen luotettavasti havainnoksi. 1287 kuvauspistettä on valikoitunut myös epätasaisesti, sillä osa kärkikolmioista on ollut esimerkiksi pitkän suoran varrella, jolloin siitä on useampia havaintoja kuin kaartavalla tiellä.

Tarkasteluun saattoi sisältyä muitakin virheitä, kuten puun tai muun kasvillisuuden aiheuttamien okkluusioiden muodossa. Kohdehavainnon kokonaan estäviä kuvia ei valikoitunut koulutukseen, joten niistä ei syntynyt vääränlaista koulutusaineistoa. Pienet okkluusiot voivat itsessään olla koulutusta tehostavia tekijöitä, jotta algoritmin tunnistuskykyä onnistutaan monipuolistaman. Tällöin reaalimaailman haasteet ovat paremmin huomioitavissa.


*Kuva 50. Erilaisia tapauksia vääränlaisesta aineistosta, joita ei voinut käyttää koulutuksessa. Ensimmäisessä kuvassa automatiikka on anonymisoinut kärkikolmion, minkä havaita toisena olevasta kuvasta, joka on otettu hetkeä myöhemmin. Tässä kärkikolmio on näkyvissä. Kolmannessa kuvassa merkki on kuvattu väärinpäin ja neljännessä etäisyys aiheuttaa haasteita.*

Tunnistamisia olisi voitu lisätä ottamalla tarkasteluun myös alemman kynnysarvon (*threshold value*) kohteita, mutta tämä olisi myös lisännyt vääriä positiivisia tunnistuksia. Testimielessä tätä kuitenkin yritettiin asettamalla tunnistuksen kynnysarvoksi 0.25, joka on varsin matala arvo. Alemmissa kuvissa 51 ja 52 on esitetty vääriä positiivisia, joita tämän myötä ilmeni. Algoritmi on tunnistanut kolmionmuotoisia kohteita, kuten kattoja sekä väriltään samantyyppisiä kohteita, kuten toisia liikennemerkkejä sekä aitoja.


*Kuva 51. Vasemmanpuoleisesta kuvassa luottamustaso on 29 ja oikeanpuoleisessa 52.*


*Kuva 52. Vasemmanpuoleinen kuva on saanut luottamustason 60 ja oikeanpuoleinen kuva arvon 78.*

Toisaalta tunnistaminen on onnistunut haastavilta kohteilta, jotka esitellään kuvissa 53 ja 54. Esimerkiksi kuvassa 53 algoritmi on onnistunut tunnistamaan kärkikolmion toisen merkin takaa, vaikka siitä on vain pieni osa havaittavissa. Samoin kuvassa 54 kärkikolmio on havaittu kasvillisuuden takaa sekä varjossa. Molemmissa tapauksissa kärkikolmio kuitenkin havaittiin korkeammalla luottamustasolla muissa kuvissa. Mikäli kuvia otetaan suhteellisen tiheästi, ei synny tarvetta alentaa tunnistustarkkuutta, koska väärät positiiviset hidastavat muuten tulosten tarkastelua.


*Kuva 53. Vasemman kuvan tunnistuksen luottamustaso oli 36 sekä oikealla sama kuva ilman tunnistusta.*

*Kuva 54.Vasemmalla olevan kuvan tunnistuksen luottamustaso on 42. Oikealla sama kuva ilman tunnistusta.*

Omassa työssä kuvataajuus oli keskiarvoisesti pilkotuilla kuvilla noin 15 ja panoraamakuvilla noin 11. Tätä nopeutta olisi voitu nopeuttaa erilaisin parametrisäädöin, mutta testauksessa aineistoa oli varsin vähän, joten tähän ei nähty tarvetta.

Kärkikolmioiden sijainnin selvityksessä tuotettujen keskipisteiden arvioitiin eroavan muutaman metrin oikeista sijainneista. Sijainnin selvittämistä olisi voitu parantaa maastokäynneillä, jotta tarkat kärkikolmiosijainnit olisi saatu selville. Tuloksiin voi myös vaikuttaa kuvausauton antamat koordinaattivirheet sekä YOLO:n tuottamien tunnistusten virheet. Maastokäynneillä myös Väyläviraston aineistoja olisi voitu tarkistaa ja saada kuvat myös väärinpäin olleista kärkikolmioista. Näistä väärinpäin olevista kärkikolmioista olisi voitu myös tehdä oma tunnistusprosessinsa, sillä muita kärkikolmion muotoisia liikennemerkkejä ei ole käytössä. Täten näiden tunnistaminen muodon perusteella olisi ollut todennäköisesti mahdollista. Toisaalta harmaansävyinen tausta olisi voinut olla hankala erottaa joistakin muista kohteista, mikä olisi mahdollisesti lisännyt vääriä tunnistuksia. Omien tulosten suoraan vertaaminen Väyläviraston aineiston kanssa ei myös ehkä ole oleellista, sillä Väyläviraston aineistossa on todennäköisemmin vain valtion hallinnoimien tieosuuksien liikennemerkit, jolloin kuntien hallinnoimilla teillä olevat merkit puuttuvat.

## 6.3 Haasteet tunnistuksessa ja konvoluutioverkostojen käytössä

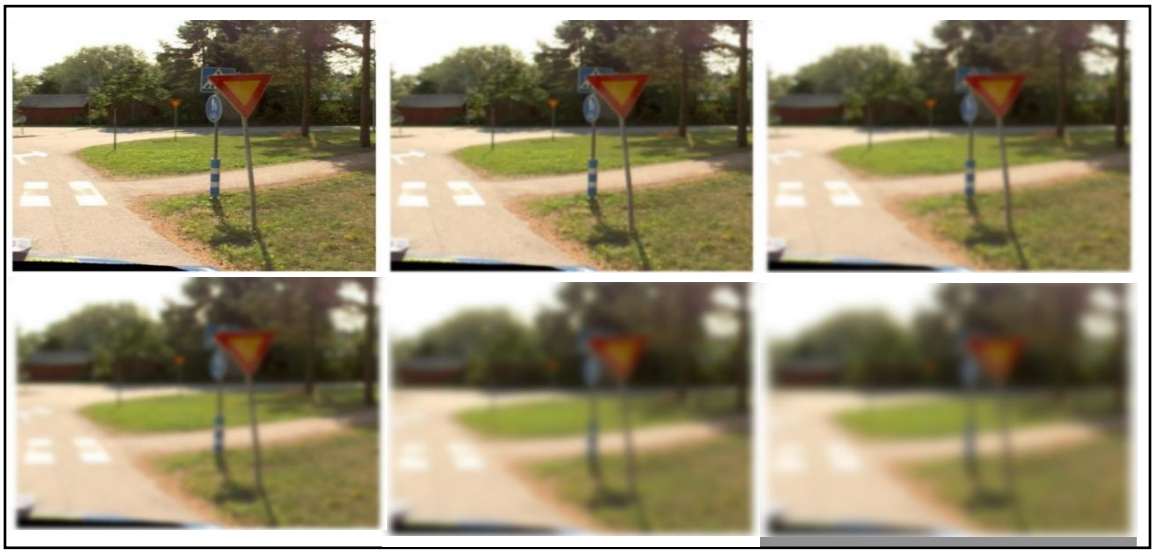
Yksi havaittu haaste tässä työssä oli laskentatehon tarve, sillä kohteentunnistusalgoritmin kouluttaminen vaatii runsaasti tehoja tietokoneelta (Arcos-García et al. 2018, Du 2018, Voulodimos et al. 2017). Toisena haasteena on se, että kaikki aineistot eivät sovellu käsittelyyn (Du 2018, Li et al. 2019). Konvoluutioverkostot tarjoavat kuitenkin tehokkaan matemaattisen tavan ilmentää ja irrottaa (*extract*) piirteitä siihen syötetystä aineistosta (Du 2018). Uudet kohteentunnistusalgoritmit pystyvät tunnistamaan jo havaitsemansa kohteet ja näin nopeuttamaan verkoston toimintaa sekä vähentämään laskentatehon tarvetta. Toisaalta runsaskohteinen kuvaustausta voi myös vaikuttaa algoritmin toimintakykyyn hidastavasti (Du 2018, Li et al. 2019, Zhang et al. 2017).

Voulodimos et al. (2017) toteavat, että olisi oleellista tehdä enemmän pohjatyötä syväoppimismenetelmien implementoinnissa. Toimivaa viitekehystä ei esimerkiksi ole vielä luotu, minkä pohjalta oikeanlaisen mallin valinta olisi optimaalista tietyn aineiston perustella. Samoin perustelut olla käyttämättä jotain menetelmää ovat puutteelliset. Tämä ongelma on havaittu myös aiemmin, esimerkiksi Stallkamp et al. (2012) pohtivat menetelmien vertailun olevan puutteellista ja jopa vääristynyttä. Algoritmien kehitystä ovat ohjanneet avoimet aineistot ja algoritmit on optimoitu testaamalla niiden suoritusta näiden avulla (Li et al. 2019).

Esimerkiksi avoimet kuvalähteet kuten PASCAL VOC -aineistot sisältävät kuvia, joissa liikennemerkit ovat 20 % kuvasta, ja monet algoritmit ovat testattu näiden aineistojen avulla (Zhu et al. 2016). Liikennemerkkien koko on panoraamoissa yleensä pieni osa kuvaa, esimerkiksi vain yksi sadasosa (Zhu 2016). Tämä johtaa siihen, että panoraamakuvien osalta algoritmien toiminta ei ole optimaalista. Myös omassa työssä havaittiin pienten kohteiden olevan erityisen vaikeita tunnistaa panoraamakuvilta. Merkkien koon ollessa muuhun kuvaan verrattuna suuri, tunnistus oli huomattavasti todennäköisempää.

Olosuhteiden tuomat hankaluudet liikennemerkkien tunnistamisessa on havaittu useassa tutkimuksessa (Hienonen 2014, Li et al. 2019, Stallkamp et al. 2012, Temel et al. 2019, Zhang et al. 2017, Zhu et al. 2016). Näitä ovat muun muassa valoisuuden vaihtelu, sään tuomat ongelmat, kuvausperspektiivin muutokset sekä osittaiset okkluusiot. Sään aiheuttamia ongelmia ovat muun muassa sade, sumu ja lumi, jotka kaikki voivat estää liikennemerkin havaitsemista (Hienonen 2014, Temel et al. 2019). Okkluusiot voivat ovat osittaisia peittäviä tekijöitä kuvissa merkkien edessä, kuten kasvillisuutta tai graffiteja (Stallkamp et al. 2012, Zhang et al. 2017). Stallkamp et al. 2012 mukaan nämä häiriöt aiheuttavat enemmän ongelmia konenäölle kuin ihmisen havainnointiin. Kuvausperspektiivin muutokset voivat riippua esimerkiksi kuvausauton nopeudesta ja kuljettajan ajotyylistä (Temel et al. 2019). Lisäksi muutokset terävyydessä (kuva 55) voivat vaikuttaa kohteentunnistuksen kykyyn.

Du (2018) lisää, että tunnistettavien kohteiden läheisyys toisiinsa heikentää myös kohteentunnistusalgoritmien toimintaa. Temel et al. (2019) myös toteavat, että algoritmien kehittämisessä olisi otettava enemmän huomioon hankalien sääolosuhteiden vaikutuksia kuviin, jotta algoritmit olisivat paremmin implementoitavissa reaalimaailmaan.



*Kuva 55. Terävyyden vaihtelu voi vaikeuttaa liikennemerkkien tunnistamista. Kuva mukailtu (Suuriniemi et al. 2019) kuvasta 2.*

## 6.4 Jatkoehdotukset

Algoritmin toimintaa olisi voinut testata eri kokoisilla kuvilla vielä lisää esimerkiksi testaamalla, missä vaiheessa panoraamojen ja niistä luotujen osakokonaisuuksien välillä tunnistustulokset ovat riittävän hyviä ja prosessointiaika mahdollisimman lyhyt. Samoin vielä pienemmistä kuvista olisi voitu saada parempia tuloksia esimerkiksi kauempaa havaituilta kärkikolmioilta. Erilaisin parametrimuutoksin GPU:n toimintaa olisi voitu saada myös nopeutettua. Valitsemalla paremmin tarkasteltavan aineiston, olisi aineiston validointia voitu myös nopeuttaa. Tällä viitataan siihen, että osa valitusta koulutus- ja validointiaineistosta piti poistaa, koska niissä oli esimerkiksi anonymisoituja kohteita tai kärkikolmio oli kuvattu väärältä puolelta. Lopulta näitä tapauksia oli onneksi varsin vähän huomioiden muuten kuva-aineiston suuren määrän.

Kuten aiemmin on mainittu, panoraamakuvilta on mahdollista mitata tarkasti havaittavien kohteiden sijainteja. Tätä voi toteuttaa esimerkiksi fotogrammetristen menetelmien avulla (Salo 2018). Mittaamalla maastossa kärkikolmioiden tarkan sijainnin olisi näitä voitu verrata tarkemmin kuin visuaalisella tarkastelulla.

Aineistoa olisi voitu myös kerätä avoimista lähteistä, joita on runsaasti saatavilla. Esimerkiksi Ruotsista löytyy paljon samankaltaisia liikennemerkkejä kuin Suomesta, joita myös löytyy avoimista aineistoista (Larsson & Felsberg 2011). Valmiin aineiston käyttö koulutusaineistona vähentäisi työmäärää ja kustannuksia. Tämä ei kuitenkaan poista testausaineiston tarvetta, joten jollain tavalla kuvausaineistoa olisi joka tapauksessa kerättävä.

# 7. Johtopäätökset

Kohteentunnistus on mahdollista erilaisilla kuva-aineistoilla. Niin panoraamakuvista kuin niistä luoduista osakokonaisuuksistakin havaittiin useita kärkikolmioita. Kohteentunnistusalgoritmien toimintaa edistää kohteen mahdollisimman suuri koko kuvassa, joten tunnistaminen onnistui paremmin osakokonaisuuksista. Täten tavoitteesta ja kuvausmateriaalista riippuen käytetty kuvanottoväli määrittää paljon. Jos halutaan tunnistaa liikennemerkki todennäköisemmin kuvajoukosta, on syytä pilkkoa kuvia pienemmiksi osakokonaisuuksiksi. Mikäli nopeus on tärkein kriteeri, eikä merkkiä ole välttämätöntä tunnistaa jokaisesta kuvauspisteestä, on mahdollista myös käyttää panoraamakuvia. Panoraamakuvat on tässä tapauksessa otettava hyvin läheltä kohdetta, sillä tunnistustaso laskee voimakkaasti etäisyyden myötä. Vääriä havaintoja (*false positive*) näyttää myös syntyvän enemmän sekä kohteen sijoittaminen kuvilla (eli missä kohde on kuvassa) on epätarkempaa panoraamakuvilta. Niiden antamien tulosten käsittelyyn kuluu todennäköisesti tämän myötä enemmän aikaa.

Kohteiden luokittelu sekä paikallistaminen kuvassa onnistui myös hyvin ja tähän lopputulokseen päästiin 700 kuvan koulutusaineistolla. Muun muassa Väyläviraston tuottamassa raportissa todetaan, että annotoitua aineistoa olisi tuotettava tuhansia yhtä kohdetta kohti (Suuriniemi et al. 2019). Tämän työn perusteella ei aineistoa tarvitse olla kuitenkaan niin paljoa hyvien tunnistustulosten mahdollistamiseksi. Lisäämällä koulutusaineiston määrää voisi kuitenkin olla mahdollista saada vielä paremmin reaalimaailmaa vastaava tunnistusalgoritmi. Kun algoritmin parametrit on valittu ja malli saatu toimivaksi, on kuva-aineistojen annotointi itsessään työläin työvaihe. Tämän vuoksi esimerkiksi Väylävirasto olisi valmis maksamaan myös valmiiksi annotoidusta aineistoista (Suuriniemi et al. 2019). On kuitenkin syytä huomata, että ilman teknistä taustaa on koneoppimisalgoritmien implementointi varsin haastavaa, mikä havaittiin myös tässä työssä.

*Bounding box* -rajaukset liikennemerkkien sijainneista kuvissa olivat varsin luotettavia ja niiden perusteella pystyttiin laskemaan kärkikolmion sijainti kuvassa. Visuaalisella tarkastelulla havaittu muutaman metrin tarkkuus ei kuitenkaan ole riittävä luotettavan sijainnin määrittelemiseksi. Täten olisi hyvä tehdä maastokäyntejä, jotta liikennemerkkien oikeat sijainnit saataisiin määritettyä. Tämän myötä oikeita sijainteja voisi verrata laskennan tuloksiin ja saada osviittaa laskennan tarkkuudesta.

Kuten aiemmin on mainittu, useassa tutkimuksessa todettiin algoritmien olevan optimoituja lähinnä pienempien kuva-aineistojen käsittelyyn ja on mielenkiintoista nähdä, onnistutaanko tulevaisuudessa kehittämään paremmin panoraamakuviin soveltuvaa algoritmia. Esimerkiksi Zhu et al. (2016) ja Li

et al. (2019) tuottamien avoimien panoraamakuva-aineistojen myötä tämän kehittäminen on kuitenkin todennäköistä. Panoraamakuviin paremmin soveltuvan algoritmin myötä kuvien pilkkomiseen kuluva prosessointiaika vähenisi.

Algoritmien koulutus vaatii joka tapauksessa paljon kuvia. Suuriniemi et al. (2019) käyvät läpi näiden keräämiseen liittyviä ongelmia. Esimerkiksi, jos koko Suomen kattavia kuvauksia ei järjestettäisi standardisoidusti vaan monen toimijan toimesta, on mahdollista, että aineistosta tulisi liian vaihtelevaa. Lisäksi jotkin alueet voisivat jäädä kuvaamatta, joten pelkästään muille toimijoille kuvausvastuuta tuskin pystyisi jättämään. Väylävirasto onkin suunnitellut keräävänsä kuva-aineistoja esimerkiksi muun perustoiminnan ohessa, kuten tienhoidon ja mittauksien yhteydessä (Suuriniemi et al. 2019).

Mikäli kuvauksia järjestettäisiin koordinoidusti kuvausautoilla, olisi huomioitava esimerkiksi Stallkamp et al. (2012) tutkimuksen havainnot, joiden mukaan kuvausauton olisi pyrittävä ajamaan tasaista ja melko hidasta vauhtia, jotta korkean nopeusrajoituksen merkit eivät olisi sumeita. Muuten hitaan nopeuden merkit (esim. STOP-merkit ja kärkikolmiot) korostuvat oikeina havaintoina. On kuitenkin mahdollista, että kuvausteknologian kehittyminen vuosien varrella on pienentänyt tätä ongelmaa.

On toisaalta täysin mahdollista, että kansalaiset tai muut toimijat voivat täydentää olemassa olevaa aineistoa. Tämän tutkimuksen mukaan voidaan arvioida myös huonompilaatuisten kuvien sopivan aineiston koulutukseen. Väylän teettämässä raportissa todetaan kuitenkin, että kansalaisille voisi olla vaikeaa tarjota kannustimia tuottaa tarkoitukseen sopivaa aineistoa (Suuriniemi et al. 2019, 19). Mahdollisuutena kuitenkin pidetään yksityishenkilöiden halua tuoda ilmi puutteet tienhoidossa, mikä voisi motivoida heitä tuottamaan aineistoa omien olosuhteidensa parantamiseksi. Tässä työssä havaittiin Mapillaryyn toimitettujen kuva-aineistojen toimivan hyvin koulutusaineistona ja nämäkin kuvat on tuotettu erilaisista lähteistä joukkoistamalla. Toisaalta kuvat olivat varsin tarkkoja eikä ole takuuta siitä, että kaikki yksityishenkilöt voisivat tuottaa yhtä laadukasta aineistoa. Tähän voisi vastata vaatimalla kuvaukselta tiettyä tasoa, jotta kuvat hyväksyttäisiin käsittelyyn.

Syväoppimisen menetelmiä on onnistuttu hyödyntämään monilla eri osa-alueilla ja niiden toiminta on tuottanut hyviä tuloksia. Liikennemerkkien tunnistuksen lisäksi esimerkiksi kasvojentunnistus on ollut suosittua konenäön parissa työskentelevien kanssa (Voulodimos 2017, 3.2, Wu et al. 2019). Liikennemerkkien tunnistuksen lisäksi esimerkiksi Hienosen (2014) työssä tutkittiin liikennemerkkien kunnon selvittämistä kohteentunnistuksen avulla. Tämän myötä tieinventaarioiden vähintään

osittainen automatisointi oli mahdollista. Myös esimerkiksi jalankulkijoiden tunnistaminen on onnistunut hyvin (Wu et al. 2019). Tulevaisuudessa konenäön ja tämän myötä kohteentunnistuksen kyvykkyyden voidaan arvioida parantuvan. On kuitenkin otettava huomioon esimerkiksi Nixon & Aguado (2019) argumentti, jonka mukaan konenäkö ei voi toimia täysin ihmisnäön vertaisesti, koska emme vieläkään ymmärrä täysin, miten aivojen ja silmien välinen yhteys toimii. Niinpä konenäkösysteemin rakenne ei voi pohjautua tähän, joten sen toiminnallisuuskaan ei voi päästä täysin ihmissilmän tasolle. Syväoppimismetodit jatkavat kuitenkin kehittymistään ja uusien käyttötarkoitusten uskotaan lisääntyvän tulevina vuosina laskentakapasiteetin ja big datan kasvaessa (Guo et al. 2020, s.380).

# 8. Kiitokset

Kiitos Maanmittauslaitokselle työn mahdollistamisesta. Suuret kiitokset mainiosta ohjauksesta Maanmittauslaitoksen Antti Jakobssonille, Juha Oksaselle ja Teemu Mieloselle sekä yliopiston ohjaaja Petteri Muukkoselle. Kiitos myös Tommi Terävälle neuvonannosta.

Special thanks to TietoEvry's Iftikhar Ahmad for helping me with the settings of the Azure environment, implementing YOLO and giving guidance throughout the thesis. I'm also grateful for the help Madelen and Muthu from Mapillary provided.

Kunniamaininnan ansaitsee Joonas Jokela, joka puski gradua eteenpäin chatin välityksellä antaen tarpeellista potkua tiukkoihin hetkiin. Lisäksi ihanaiset kiitokset Venlalle oikoluvusta ja tuesta. Perhe ja kaverit, teitä pitäisi myös ilmeisesti kiittää. Kiitos mitättömästä panoksesta kirjoitustyöhön, mutta suurenmoisesta avusta elämässä.

Toivottavasti kärkikolmiot häipyvät vihdoin painajaisistani.

# Lähteet


Aleju (2020). ImgAug – Github. <https://github.com/aleju/imgaug> Viitattu: 18.3.2020

Alexey AB (2020). Darknet – Github. <https://github.com/AlexeyAB/darknet> Viitattu: 18.3.2020

Allen, C., Tsou, M., Aslam, A., Nagel, A., & Gawron, J. (2016). Applying GIS and Machine Learning Methods to Twitter Data for Multiscale Surveillance of Influenza. PLoS One. 2016; 11(7). <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4959719/>

Arcos-García, Á., Alvarez-Garcia, J. & Soria-Morillo, L. (2018). Evaluation of deep neural networks for traffic sign detection systems. *Neurocomputing* 316, 332–344. <https://www.sciencedirect.com/science/article/pii/S092523121830924X>

Ball, J. Anderson, D. & Chan, C. (2017). A Comprehensive survey of deep learning in remote rensing: Theories, tools and challenges for the community. *Journal of Applied Remote Sensing*. 11(4), 1–64. <https://arxiv.org/abs/1709.00308>

Du, J. (2018). Understanding of object detection based on CNN family and YOLO. *Journal of Physics*. Conf. Ser. 1004 012029, 1–8. <https://iopscience.iop.org/article/10.1088/1742-6596/1004/1/012029/pdf>

Garg, P. & Chowdhury, D. & More, V. (2019). Traffic sign recognition and classification using YOLOv2, Faster RCNN and SSD. 1–5. Conf. Ser.10.1109/ICCCNT45670.2019.8944491. <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8944491>

Garosi, Y., Sheklabadi, M., Conoscenti, C., Pourghasemi, H., & Van Oost, K. (2019). Assessing the performance of GIS- based machine learning models with different accuracy measures for determining susceptibility to gully erosion. *Science of The Total Encironment.* 664, 1117–1132. <https://www.sciencedirect.com/science/article/pii/S0048969719305741?>

Geotrim esittelymateriaali (2018). Trimblen mobiilikartoitusportfolio. <https://docplayer.fi/108097569-Trimblen-mobiilikartoitusportfolio-trimble-mx9-trimble-mx2-trimble-mx7.html>

Guo, H., Goodchild, M. & Annoni, A. (2020). *Manual of Digital Earth*, 846. Singapore: Springer. <https://link.springer.com/content/pdf/10.1007%2F978-981-32-9915-3.pdf>

Hienonen, P. (2014). *Automatic traffic sign inventory- and condition analysis*. Master's Thesis Lappeenranta University of Technology, School of Industrial Engineering and Management, s. 91. <https://lutpub.lut.fi/bitstream/handle/10024/98984/thesis.pdf>

Ju, M., Luo, H., Wang, Z., Hui, B. & Chang, Z. (2019). The Application of improved YOLO V3 in multi-scale target detection. *Applied Sciences*. 9(18):3775.

Kanevski, M., Foresti, L., Kaiser, C., Pozdnoukhov, A., Timonin, V. & Tuia, D. (2009). Machine learning models for geospatial data. *Handbook of Theoretical and Quantitative Geography*, 457. <https://serval.unil.ch/resource/serval:BIB_05FBCD424B3F.P001/REF.pdf>

Kauhanen, H., & Rönnholm, P. (2012). Image acquisition constraints for panoramic frame camera imaging. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. Melbourne, 397–402. <https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XXXIX-B3/397/2012/isprsarchives-XXXIX-B3-397-2012.pdf>

Khan, G., Tariq, Z.M., & Khan, M.U. (2019). Multi-Person tracking based on Faster R-CNN and deep appearance features. *Visual Object Tracking in the Deep Neural Networks Era*, Julkaisematon kirja.

Kuntahaastattelujen yhteenveto (28.2.2019). Uudet kuvanmittausteknologiat-projekti. Maanmittauslaitos.

Laki uudesta tieliikennelaista (729/2018). Finlex. <https://www.finlex.fi/fi/laki/alkup/2018/20180729#Pidp448014864>

Larsson F. & Felsberg M. (2011). Using fourier descriptors and spatial models for traffic sign recognition, in Proc. SCIA, Berlin, Heidelberg, SCIA'11, Springer-Verlag, 238–249. <https://link.springer.com/content/pdf/10.1007%2F978-3-642-21227-7_23.pdf>

Lehto, M., Neittaanmäki, P., Niinimäki, E., Nyrhinen R., Ojalainen A., Pölönen I., Rautatiainen I., Ruohonen, T., Tuominen, H. Vähäkainu, P., Äyrämö, S & Äyrämö S-M. (2019). *Tekoälyn perusteita ja sovelluksia*. <https://tim.jyu.fi/view/kurssit/tie/tiep1000/tekoalyn-sovelluksetset/kirja>

Leppänen, R. (2019). *Anomalioiden havaitseminen langattomissa sensoriverkoissa syväoppimisen avulla*. Pro gradu -tutkielma Jyväskylän yliopisto, Informaatioteknologian tiedekunta, 112. <https://jyx.jyu.fi/bitstream/handle/123456789/66794/URN%3aNBN%3afi%3ajyu-201912135267.pdf>

Li, K., Wan, G., Cheng, G., Meng, L. & Han, J. (2020). Object detection in optical remote sensing images: A survey and a new benchmark. *ISPRS Journal of Photogrammetry and Remote Sensing*, 159, 296–307. <https://www.sciencedirect.com/science/article/pii/S0924271619302825?>

Li Y, Tong G, Gao H, Wang Y, Zhang L, Chen H. (2019). Pano-RSOD: A Dataset and Benchmark for Panoramic Road Scene Object Detection. *Electronics*, 8(3):329. <https://www.mdpi.com/2079-9292/8/3/329>

Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., & Pietikäinen, M. (2019). Deep learning for generic object detection: A survey. *International Journal of Computer Vision*, 128(2), 261–318. <https://arxiv.org/abs/1809.02165>

Lv, Q., Dou, Y., Niu, X., Xu, J., & Li, B. (2014). Classification of land cover based on deep belief networks using polarimetric RADARSAT-2 data. 2014 IEEE Geoscience And Remote Sensing Symposium. <https://ieeexplore.ieee.org/abstract/document/6947537>

Maanmittauslaitoksen taustakartta, Maanmittauslaitos. taustakarttaa käytetty karttojen taustana 1.1.2020 – 20.04.2020.

Mapillary. <https://www.mapillary.com/developer/api-documentation/#traffic-signs> Viitattu: 9.4.2020

Meng, Zibo & Fan, Xiaochuan & Chen, Xin & Chen, Min & Tong, Yan. (2017). Detecting small signs from large images. Conference Paper, IEEE International Conference on Information Reuse and Integration. 10.1109/IRI.2017.57, 1–8. <https://arxiv.org/abs/1706.08574>

Mojaddadi, H., Pradhan, B., Nampak, H., Ahmad, N. & Ghazali, A. (2017). Ensemble machine-learning-based geospatial approach for flood risk assessment using multi-sensor remote-sensing data and GIS. *Geomatics, Natural Hazards and Risk*, 2017, 8 (2), 1080–1102. <https://www.tandfonline.com/doi/full/10.1080/19475705.2017.1294113>

Nixon, M. & Aguado A. (2019). *Feature Extraction and Image Processing for Computer Vision*, 4th Edition. Imprint: Academic Press, 650. <https://www.elsevier.com/books/feature-extraction-and-image-processing-for-computer-vision/nixon/978-0-12-814976-8>

Nvidia (2016). What is the difference between deep learning training and inference? <https://blogs.nvidia.com/blog/2016/08/22/difference-deep-learning-training-inference-ai/> Viitattu: 18.3.2020

Nodari, A., Vanetti, M., & Gallo, I. (2012). *Digital privacy: Replacing pedestrians from google street view images*. In Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), Tsukuba, Japan, 2889–2893. <https://projet.liris.cnrs.fr/imagine/pub/proceedings/ICPR-2012/media/files/1786.pdf>

Pham, B., Tien Bui, D., Prakash, I., & Dholakia, M. (2017). Hybrid integration of Multilayer Perceptron Neural Networks and machine learning ensembles for landslide susceptibility assessment at Himalayan area (India) using GIS. *Catena*, 149, 52-63.<https://www.sciencedirect.com/science/article/pii/S034181621630368X?>

Putra, M. Yussof, Z. Lim, K. & Salim, S. (2018). Convolutional neural network for person and car detection using YOLO framework. *Journal of Telecommunication, Electronic and Computer Engineering* 10 No. 1–7, 1–5. <https://pdfs.semanticscholar.org/a735/3ff60115d2f28c1282d7c2ff733f3bdcc7e1.pdf>

Redmon J., Divvala S., Girshick R. & Farhadi A. (2016). *You Only Look Once: Unified, real-time object detection*. University of Washington, Allen Institute for AI, Facebook AI Research, 1–10. <https://arxiv.org/pdf/1506.02640v5.pdf>

Rosebrock A. (2016). *Intersection over Union (IoU) for object detection.* <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/> Viitattu 27.3.2020

Salo J. (2018). *Mittatarkat panoraamakuvat väyläomaisuuden hallinnan työkaluna*. Aalto-yliopisto. Insinööritieteiden korkeakoulu, 63. <https://aaltodoc.aalto.fi/handle/123456789/35537>

Saleh, S., Khwandah, S., Heller, A., Hardt, W. & Mumtaz, Ans. (2019). Traffic signs recognition and distance estimation using a monocular camera. Conference paper: Actual Problems of System and Software Engineering (APSSE 2019), Moscow, Russia, 1–13.

Stallkamp J., Schlipsing, M., Salmen, J. & Igel, C. (2011). The German traffic sign recognition benchmark: A multi-class classification competition. *Proceedings of the International Joint Conference on Neural Networks*. 1453–1460. <https://www.ini.rub.de/upload/file/1470692848_f03494010c16c36bab9e/StallkampEtAl_GTSRB_IJCNN2011.pdf>

Stallkamp, J., Schlipsing, M., Salmen, J., & Igel, C. (2012). Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, *32*, 323–332. <https://www.sciencedirect.com/science/article/pii/S0893608012000457>

Suuriniemi, S., Kettunen, L., Huuskonen, O., Halme, J-M., Kuusela, R., Myllärinen, J. & Kinnunen, P (2019). Konenäön vakiintuva hyödyntäminen tieomaisuuden hallinnassa. Väyläviraston julkaisuja 13/2019. <https://julkaisut.vayla.fi/pdf12/vj_2019-13_konenaon_vakiintuva_web.pdf>

Stenroos, O. (2017). *Object detection from images using convolutional neural networks*. Aalto University School of Science, Master's Thesis, 75. <https://aaltodoc.aalto.fi/bitstream/handle/123456789/27960/master_Stenroos_Olavi_2017.pdf>

Stewart, M. (2019). Advanced topics in deep convolutional neural networks. <https://mc.ai/advanced-topics-in-deep-convolutional-neural-networks/> Viitattu: 9.4.2020

Temel, D., Alshawi, T., Chen, M. & AlRegib, G. (2019). Challenging environments for traffic sign detection: Reliability assessment under inclement conditions, 1–15.

Terävä, T. (2020). Suullinen tiedonanto. Helsinki: Esri Finland. (31.3.2020)

Velhonoja P. & Karhunen M. (2003). *Yleisohjeet liikennemerkkien käytöstä*, 951-726-979-X. Helsinki: Tiehallinto, 2003. <https://julkaisut.vayla.fi/thohje/pdf/2000006-v-03liikennemerkkiohje.pdf>

Voulodimos A., Doulamis N., Doulamis A. & Protopapadakis E. (2018). Deep learning for computer vision: A brief review. *Computational Intelligence and Neuroscience*. 2018, 1–13.

Zhang J, Huang M, Jin X. & Li X. (2017). A Real-Time Chinese traffic sign detection algorithm based on modified YOLOv2. *Algorithms*. 2017; 10(4), 1–13. <https://www.mdpi.com/1999-4893/10/4/127>

Zhu, Z., Liang D., Zhang, S., Huang, X., Li, B. & Hu, S. (2016). Traffic-Sign detection and classification in the wild. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1–10. <https://zpascal.net/cvpr2016/Zhu_Traffic-Sign_Detection_and_CVPR_2016_paper.pdf>

Wu, X., Sahoo, D., & Hoi, S. (2019). Recent advances in deep learning for object detection. *Neurocomputing*, 1–40. <https://arxiv.org/pdf/1908.03673.pdf>

# Liitteet

Liite 1. Patches.py, Skriptin avulla voidaan pilkkoa kuvista pienempiä. Kuvakaappaukset Jupyter Notebook:sta.

```
# Inspired by https://stackoverflow.com/questions/53501331/crop-entire-image-with-the-same-cropping-size-with-pil-in-python
# and https://stackoverflow.com/questions/5953373/how-to-split-image-into-multiple-pieces-in-python

# Import needed libraries
import os
import glob
from PIL import Image
Image.MAX_IMAGE_PIXELS = None # to avoid image size warning

# Set image locations
imgdir = "E:\Panobatches\Mapillarytest"

# if you want file of a specific extension (.jpg):
filelist = [f for f in glob.glob(imgdir + "**/*.jpg", recursive=True)]
# Save directory
savedir = "E:\Panobatches\Mapillarytest\mapillaryoutput"

# Of which point should the patching start
start_pos = start_x, start_y = (0, 0)
# Image size
cropped_image_size = w, h = (832, 832)

# For each image on image list, open, set size
for file in filelist:
    img = Image.open(file)
    width, height = img.size

    frame_num = 1
    for col_i in range(0, width, w):
        for row_i in range(0, height, h):
            crop = img.crop((col_i, row_i, col_i + w, row_i + h))
            name = os.path.basename(file)
            name = os.path.splitext(name)[0]
            save_to= os.path.join(savedir, name+"_{:03}.jpg")
            crop.save(save_to.format(frame_num))
            frame_num += 1
```

Liite 2. IoU_function.py, Skriptin avulla voidaan selvittää leikkauksen ja unionin osamäärä (IoU, Intersection over Union) tunnistettujen koordinaattien avulla. Kuvakaappaukset Jupyter Notebook:sta.

```
# Inspired by https://stackoverflow.com/questions/28723670/intersection-over-union-between-two-detections
# import the necessary packages
from collections import namedtuple
import numpy as np
import cv2

# define the `Detection` object
Detection = namedtuple("Detection", ["image_path", "gt", "pred"])

# ==========================================================================

#BOX A is yolo prediction
#BOX B is XML

boxA = [779,284,795,296]
boxB = [780,284,792,296]


Detection(r"C:\Users\ollir\Desktop\inkoo\pano_0009_000013_023.jpg", [779,284,795,296], [780,284,792,296]),

examples = [
    Detection(r"C:\Users\ollir\Desktop\inkoo\pano_0009_000014_023.jpg", [779,284,795,296], [780,284,792,296])]
```

```
def bb_intersection_over_union(boxA, boxB):
    # determine the (x, y)-coordinates of the intersection rectangle
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])

    # compute the area of intersection rectangle
    interArea = (xB - xA) * (yB - yA)

    # compute the area of both the prediction and ground-truth
    # rectangles
    boxAArea = (boxA[2] - boxA[0]) * (boxA[3] - boxA[1])
    boxBArea = (boxB[2] - boxB[0]) * (boxB[3] - boxB[1])

    # compute the intersection over union by taking the intersection
    # area and dividing it by the sum of prediction + ground-truth
    # areas - the interesection area
    iou = interArea / float(boxAArea + boxBArea - interArea)

    # return the intersection over union value
    return iou
```

.

```
bb_intersection_over_union(boxA, boxB)


# loop over the example detections
for detection in examples:
    # load the image
    image = cv2.imread(detection.image_path)
    # draw the ground-truth bounding box along with the predicted
    # bounding box
    cv2.rectangle(image, tuple(detection.gt[:2]),
        tuple(detection.gt[2:]), (0, 255, 0), 2)
    cv2.rectangle(image, tuple(detection.pred[:2]),
        tuple(detection.pred[2:]), (0, 0, 255), 2)
    # compute the intersection over union and display it
    iou = bb_intersection_over_union(detection.gt, detection.pred)
    cv2.putText(image, "IoU: {:.4f}".format(iou), (10, 30),
        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
    print("{}: {:.4f}".format(detection.image_path, iou))
    # show the output image
    cv2.imshow("Image", image)
    cv2.waitKey(0)
```

Liite 3. XML_to_MAP.ipynb, Skiptin avulla voidaan selvittää YOLO:n antamien tunnistuksen ja kuvanottopaikkojen koordinaattien avulla kärkikolmioiden sijainti. Vaatii toimiakseen ArcGIS Pro 2.5. Kuvakaappaukset Jupyter Notebook:sta.

```
import arcpy
import pandas
import os
from xml.dom import minidom


def getXmlValueAsInt(data, tag):
    ret = []

    for elem in data.getElementsByTagName(tag):
        ret.append(int(elem.firstChild.nodeValue))

    return ret


def distanceToObject(imageHeight, objectHeight):
    # focalSensorRatio is the ratio of focal length of the lens (mm) to the sensor height (mm)
    # This affects the slope of the distance function. If unknown, we can estimate it based on
    # known distance of object and the result given by this function.
    focalSensorRatio = 0.2

    signWidth = 900  # Yield sign is equilateral triangle with size length of 900mm
    signHeight = (900 ** 2 - 450 ** 2) ** .5

    distance = focalSensorRatio * signHeight * imageHeight / objectHeight / 1000.0  # Convert mm to meter
    distance = (distance / 0.2827) ** 0.7775  # Error correction

    return distance

# panorama.csv must be converted to an xlsx-file. Required columns are:
# panorama_file_name - Has to be first column. Used for index.
# latitude[deg]
# longitude[deg]
# heading[deg]
detailsFile = r'C:\Users\tommi\Documents\ArcGIS\Projects\Kyltit\panorama.xlsx'
details = pandas.read_excel(detailsFile, index_col=0)

# location of xml-files of detected objects
pathYield = r'C:\Users\tommi\Documents\ArcGIS\Projects\Kyltit\yield'
fileNamesNoExt = set()

# Output geodatabase. Must exist
fgdbPath = r'C:\Users\tommi\Documents\ArcGIS\Projects\Kyltit\Kyltit.gdb'

# Output Feature Class names
detectedObjectsFC = 'DetectedObjects'
clustersFC = 'Clusters'
centersFC = 'Centers'

for root, dirs, files in os.walk(pathYield):
    if root != pathYield:
        continue

    for file in files:
        fileNamesNoExt.add(file.split('.')[0])
```

```
pointsList = []  # List containing point geometries of detected objects
for file in fileNamesNoExt:
    xmlFile = fr'{pathYield}\{file}.xml'
    xmlData = minidom.parse(xmlFile)

    det = details.loc[file, : ]

    # Location of camera
    startPoint = arcpy.Point(det['longitude[deg]'], det['latitude[deg]'])
    pointGeom = arcpy.PointGeometry(startPoint, arcpy.SpatialReference(4326))  # WGS84
    pointHeading = float(det['heading[deg]'])

    imageWidth = getXmlValueAsInt(xmlData, 'width')[0]
    imageHeight = getXmlValueAsInt(xmlData, 'height')[0]

    # Same file can have multiple detected objects
    for i in range(len(getXmlValueAsInt(xmlData, 'xmin'))):
        # Bounding box
        yieldXmin = getXmlValueAsInt(xmlData, 'xmin')[i]
        yieldXmax = getXmlValueAsInt(xmlData, 'xmax')[i]
        yieldYmin = getXmlValueAsInt(xmlData, 'ymin')[i]
        yieldYmax = getXmlValueAsInt(xmlData, 'ymax')[i]
        # zero angle is north below
        newHeading = ((yieldXmin + yieldXmax) / 2 / imageWidth * 360 - 180 + pointHeading) % 360
        newDistance = distanceToObject(imageHeight, yieldYmax - yieldYmin)

        # Discarding detections from far distance allows tighter clustering.
        # Error seems to start growing past 50 meters.

        # if newDistance > 50.0:
            # continue

        newPoint = pointGeom.pointFromAngleAndDistance(newHeading, newDistance)
        pointsList.append((newPoint, file))

print('Number of points:', len(pointsList))
```

```
fc = fr'{fgdbPath}\{detectedObjectsFC}'
arcpy.management.Delete(fc)  # Delete old Feature Class if exists

arcpy.management.CreateFeatureclass(
    fgdbPath,
    detectedObjectsFC,
    'POINT',
    None, None, None,
    arcpy.SpatialReference(3067)  # EUREF FIN TM35FIN
)

arcpy.management.AddField(fc, 'imageName', 'TEXT')  # Add source filename as an attribute

cursor = arcpy.da.InsertCursor(fc, ["SHAPE@", 'imageName'])

for point, imageName in pointsList:
    cursor.insertRow([point.projectAs(arcpy.SpatialReference(3067)), imageName])

del cursor
```

```
clusterLayer = fr"{fgdbPath}\{clustersFC}"
arcpy.management.Delete(clusterLayer)

arcpy.stats.DensityBasedClustering(
    fc,
    clusterLayer,
    "DBSCAN",
    3,  # Number of features required to form a cluster. Less are considered noise
    "5 Meters",  # Search radius
    None
)

# Delete cluster containing noise features
arcpy.management.MakeFeatureLayer(clusterLayer, 'tempLayer')
arcpy.management.SelectLayerByAttribute('tempLayer', 'NEW_SELECTION', 'CLUSTER_ID = -1')

if int(arcpy.management.GetCount('tempLayer').getOutput(0)) > 0:
    arcpy.management.DeleteFeatures('tempLayer')

arcpy.management.Delete('tempLayer')
```

```
centersLayer = fr"{fgdbPath}\{centersFC}"
arcpy.management.Delete(centersLayer)

arcpy.stats.MeanCenter(
    clusterLayer,
    centersLayer,
    None,
    "CLUSTER_ID",
    None
)
```

Liite 4. Muokattu konfiguraatiotiedosto (*CFG-file*), jolla määritetään parametrit kohteentunnistajan koulutukseen.

```
[net]
# Testing
 batch=64
 subdivisions=16
# Training
# batch=64
# subdivisions=8
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 4000
policy=steps
steps=3200,3600
scales=.1,.1

 [convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky

# Downsample

[convolutional]
batch_normalize=1
filters=64
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=32
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=leaky

[shortcut]
```

```
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=128
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=256
size=3
stride=2
pad=1
activation=leaky
```

```
[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
```

```
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
```

```
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=512
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
```

```
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
```

```
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
```

```
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
```

```
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

######################

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
```

```
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear

[yolo]
mask = 6,7,8
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=1
num=9
jitter=.3
ignore_thresh = .5
truth_thresh = 1
random=1

[route]
layers = -4
```

```
[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[upsample]
stride=2

[route]
layers = -1, 61

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky
```

```
[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear

[yolo]
mask = 3,4,5
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=1
num=9
jitter=.3
ignore_thresh = .5
truth_thresh = 1
random=1

[route]
layers = -4

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[upsample]
stride=2

[route]
layers = -1, 36

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
```

```
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear

[yolo]
mask = 0,1,2
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=1
num=9
jitter=.3
ignore_thresh = .5
truth_thresh = 1
random=1
```